Hakobyan Howhannes

Batch Size and Obstacle Avoidance:

A PPO-Based Study of Autonomous Vehicles Using Unreal Engine Learning Agents

Supervisor: Boury Fries

Coach: Geeroms Kasper

Graduation Work 2024-2025

Digital Arts and Entertainment

Howest.be

1. CONTENTS

2. ABSTRACT & KEY WORDS

(EN)

This research investigates how policy batch size influences an autonomous vehicle's static obstacle avoidance success rate using Proximal Policy Optimization (PPO) in Unreal Engine's Learning Agents 5.4. The study explores five batch sizes: 16, 32, 64, 128, and 256. Their impact on obstacle avoidance is evaluated on a linear, circular, and zigzag racing track. After training each batch size for 5000 iterations, the performance is measured. Results show that larger batch sizes (128 and 256) generally achieve higher success rates. However, the performance varies across tracks. The findings suggest that while larger batch sizes stabilize training, outcomes depend on track characteristics and training configurations. This study contributes to understanding the tradeoffs in policy batch size selection and demonstrates the potential of Learning Agents for autonomous driving research.

(NL)

In dit onderzoek wordt onderzocht hoe de policy batchgrootte van invloed is op het succespercentage van een autonoom voertuig bij het vermijden van statische obstakels met behulp van Proximal Policy Optimization (PPO) in Unreal Engine Learning Agents 5.4. De studie onderzoekt vijf batchgroottes: 16, 32, 64, 128 en 256. Hun invloed op het vermijden van obstakels wordt geëvalueerd op een lineaire, cirkelvormige en zigzag racebaan. Na het trainen van elke batchgrootte gedurende 5000 iteraties worden de prestaties gemeten. De resultaten tonen aan dat grotere batchgroottes (128 en 256) over het algemeen hogere succespercentages bereiken. De prestaties variëren echter per racebaan. De bevindingen suggereren dat grotere batchgroottes de training stabiliseren, maar dat de resultaten afhankelijk zijn van de baankenmerken en trainingsconfiguraties. Deze studie draagt bij aan het begrijpen van de afwegingen bij de selectie van de batchgrootte en toont het potentieel aan van Learning Agents voor onderzoek naar autonoom rijden.

Keywords: Autonomous Driving, Reinforcement Learning, Obstacle Avoidance, Unreal Engine, Learning Agents, policy batch size, Proximal Policy Optimization

3. PREFACE

I started this research because I enjoy challenging myself to work on something intellectually stimulating and fun. As I looked into different topics, machine learning stood out to me. This is primarily because I have no prior knowledge of the topic, and I am very enthusiastic about exploring new areas of game development. Looking into autonomous driving was an ideal fit, coupled with my curiosity about Unreal Engine's Learning Agents plugin.

Learning agents is a new addition to Unreal Engine and it lacks the wealth of online resources typically available for more established tools. This inspired me to dive into uncharted territory and contribute to a growing field. I have always enjoyed working with new technology, as intimidating as it sounds.

I hope this work not only illuminates the potential of Unreal Engine’s Learning Agents but also inspires others to explore new technologies and embrace the challenges they present.

4. LIST OF FIGURES

5. INTRODUCTION

Autonomous driving has been a hot topic in the last decade due to the development of artificial intelligence (AI) and reinforcement learning (RL) algorithms. Various RL algorithms allow vehicles to drive in unknown and complex environments without human intervention, which opens endless possibilities in industries such as robotics and transportation. One interesting use case is autonomous driving in games. Developing an AI that can learn to drive and adapt its behavior to new environments without needing explicit programming takes AI programming to a whole new level. Achieving such behavior is complex and requires careful RL parameter tuning.

Various parameters in RL affect the resulting autonomous driving vehicle. One of the parameters is the policy batch size, which determines how many samples of past experiences are processed during policy updates. Previous research has focused on neural network architecture, reward design, environmental complexity, etc. The relationship between policy batch size and autonomous driving is underrepresented in the scientific community, which motivates this study. This study aims to discover how the policy batch size affects an autonomous vehicle's static obstacle avoidance rate. Unreal Engine is paired with the Learning Agents plugin to achieve this. The vehicle's behavior is guided by Proximal Policy Optimization (PPO).

This research aims to:

- Contribute to the growing body of knowledge on autonomous driving
- Explore the practicalities of using Unreal Engine's Learning Agents for machine learning

To conduct this research, the following research question was formulated:

How does policy batch size influence the static obstacle avoidance success rate, defined as the percentage of obstacles avoided, for an autonomous vehicle trained with Proximal Policy Optimization (PPO) in a simulation using Unreal Engine's Learning Agents?

To address this question, the following hypotheses were developed:

1. **There exists an optimal policy batch size greater than the size of 64 that maximizes the obstacle avoidance success rate.**
2. **Larger policy batch sizes (e.g., 128 or 256) will result in a higher obstacle avoidance success rate than smaller batch sizes (e.g., 16, 32, or 64).**

The next section reviews relevant literature on reinforcement learning, batch size, and autonomous driving. Afterward, a detailed description of the methodology, including the experimental setup and training procedures, is presented. The results are then presented and analyzed in the context of the hypotheses.

6. LITERATURE STUDY / THEORETICAL FRAMEWORK

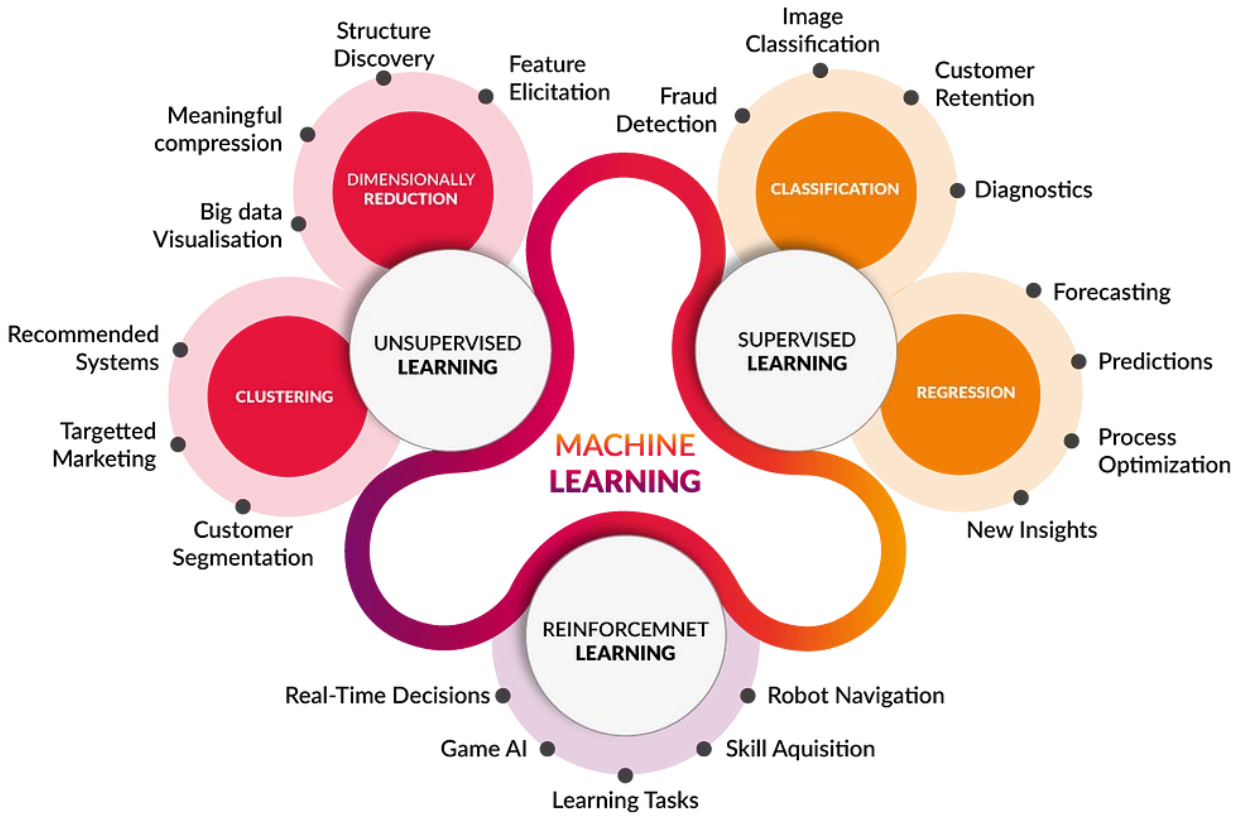
This section provides an overview of existing research and theoretical concepts relevant to the impact of experience replay batch size and obstacle avoidance AVs trained using PPO. It establishes the background needed to contextualize this study.

6.1 MACHINE LEARNING

Machine learning (ML) enables computers to perform actions without explicit programming (Bi et al., 2019). When complex behaviors are required, explicitly programming them is often burdensome. ML provides a more efficient approach to developing advanced, human-like behaviors (Janiesch et al., 2021).

There are several types of ML, such as Supervised ML, Unsupervised ML, Reinforcement Learning (RL), and Semi-Supervised ML (Jain, 2023). This paper will only explore RL, as it is a common choice when training AVs (Kiran et al., 2021).

Figure 1: Different types of ML (Experfy, n.d.)



6.2 REINFORCEMENT LEARNING (RL)

Reinforcement Learning (RL) is an area of machine learning where the agent develops its behavior based on actions, observations, and rewards. A common way of achieving this is by trial and error. The agent is forced to interact with the environment, and based on its actions, a positive or negative reward, also referred to as a penalty, is given. The goal is to maximize the total reward while balancing exploration (trying new actions) and exploitation (choosing the best-known and most rewarding action) (Crespo & Wichert, 2020).

The environment in which the agent is being trained can be extensive, with numerous observable elements and a wide range of possible actions. The developer is responsible for specifying what an agent observes, the actions it can take, and the rewards for specific behaviors.

Here are some key concepts in RL:

- *The State* represents the current condition of the environment. It encompasses all the information about the world (OpenAI, 2018).
- *Observations* represent the information the agent receives from the environment. They are used to capture the current state of the environment. This information is then added to the agent's knowledge base. Observations in a traffic situation might include the positions, directions, and speeds of nearby vehicles or the weather conditions. These observations heavily influence the agent’s decision-making (Arabaci, 2023).
- *Actions* represent the interactions with the environment. The collection of all valid actions in an environment is called the **action space** (OpenAI, 2018). One of the main challenges is finding the perfect balance between actions that yield high results and actions that have not been executed yet as part of the discovery process (Kiran et al., 2021). Revisiting the earlier traffic example, possible actions could include throttle, braking, and steering.
- *Policies* are the link between the current state of the agent's environment and the action it should take in that situation. It can be a function, a matrix, or an array of probabilities of each action being chosen (Crespo & Wichert, 2020). The words “agent” and “policy” are often used interchangeably because the policy is essentially the agent’s brain (OpenAI, 2018).
- *Rewards* play a significant role in the agent’s development. The agent receives a reward from the environment depending on its actions. If that action leads to desired outcomes, the reward is positive; otherwise, the reward is negative. The agent strives to maximize the cumulative rewards received throughout its training, leading to a desired behavior (Kiran et al., 2021). In the previously mentioned traffic scenario, driving through a red light or crashing into other cars could yield negative rewards, while driving under the speed limit and avoiding collisions could lead to positive rewards.

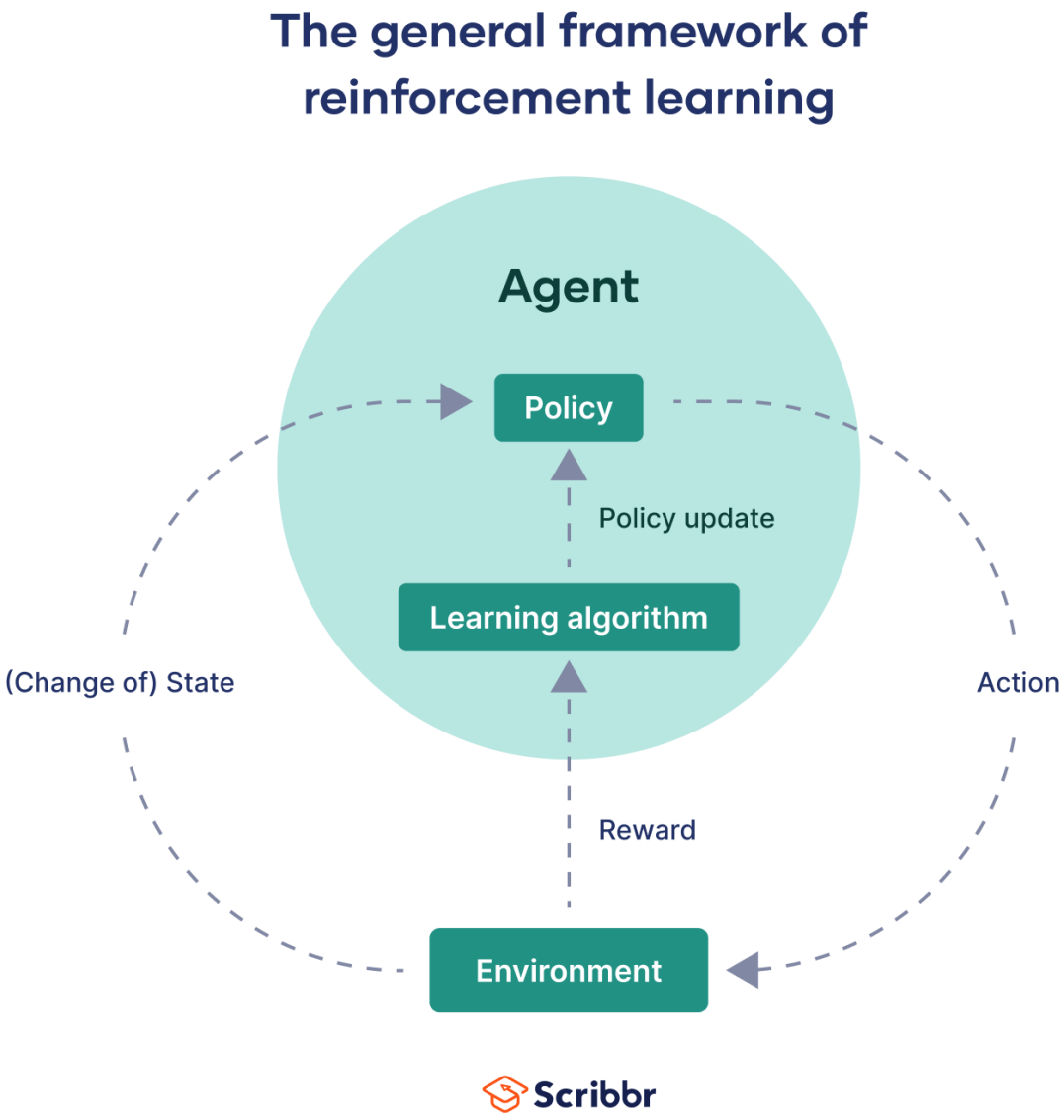
As previously mentioned, reward shaping is crucial in RL. A well-designed reward function will lead to a more desirable outcome, while a poorly designed one might lead to suboptimal behavior. To shape the reward function properly, one must define the agent's goal and the key behaviors the agent must learn. Afterward, all the actions or states that reward or penalize the agent must be identified.

- **Rewards** are given to encourage the agent to take certain actions in specific scenarios
- **Penalties** are given to discourage the agent from taking incorrect actions.

It is also important to note that having disproportionately high or low rewards or penalties can lead the agent to over-focus on specific actions, not prioritizing reaching the goal (GeeksforGeeks, 2024).

In RL, **cumulative reward**, or the **return**, is used instead of immediate reward. It is the total reward accumulated over time. The agent's goal is to maximize the return (GeeksforGeeks, 2024).

Figure 2: Reinforcement learning framework (Scribbr, n.d.)

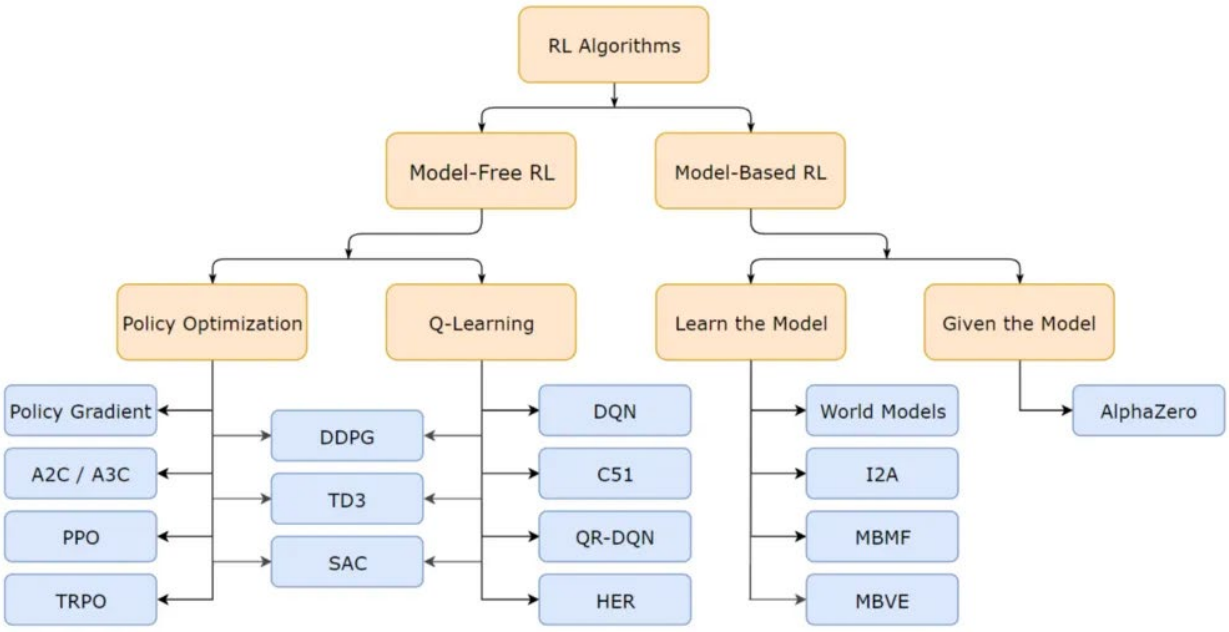


6.3 PROXIMAL POLICY OPTIMIZATION

Proximal Policy Optimization (PPO) is one of many RL algorithms developed by John Schulman in 2017. It is widely used by the American artificial intelligence company OpenAI and is considered a state-of-the-art RL algorithm by many experts (Kumar, 2024; OpenAI, 2017).

PPO is part of policy optimization methods, where the agent learns the **policy function** that maps state to action directly (SmartLab, 2019). The policy function must be evaluated to determine how well an agent performs and which action the agent should take for each state. When the agent is in the initial stages of learning and does not know which actions lead to the best results, it determines this by calculating the **policy gradients.** The output of the gradients helps the agent determine which actions are most likely to lead to higher rewards (Kumar, 2024).

Figure 3: Reinforcement learning taxonomy (OpenAI, 2018)



6.4 POLICY BATCH SIZE

Policy batch size refers to the number of samples processed together during training. In the context of RL, it specifically denotes the number of experiences grouped for training updates. Each experience typically comprises:

- The **state** of the agent
- The **action** the agent takes in that state
- The **reward** of the set action
- The **next state** of the agent
- Whether the episode terminates (Bacancy, 2024).

A batch is processed before calculating a gradient update. The developer controls the size of this batch. For performance reasons, it is advisable to keep the batch size as a power of two (jcm69, 2017).

Batch size is an important hyperparameter to tune. A batch size that is too large will lead to poor generalization, and a smaller one is not guaranteed to deliver the desired outcome. It is also known that a smaller batch size leads

to faster training, while a relatively large one leads to “guaranteed” results. It is important to mention that making general statements about the effects of hyperparameters is very difficult, as behavior often varies from dataset to dataset (Shen, 2018). Therefore, it is up to the developer to determine what is referred to as the “optimal batch size” for the training process.

6.5 AUTONOMOUS DRIVING

Autonomous driving (AD) is the process of navigating a vehicle with partial or no human intervention. AVs utilize technology to avoid road hazards and adapt to traffic conditions (CFSS, n.d.). It has been a popular research topic since the end of the last century because it promises many benefits such as safety and time savings (Wen et al., 2020).

Thanks to the rapid development of hardware, simulations enable us to test various AD algorithms with significantly reduced labor costs and time. More importantly, it takes the human error of breaking expensive equipment out of the equation (Wen et al., 2020).

Decision-making systems for real-life AV are often complex, including various sensors to fetch data from the environment, mapping algorithms for localization, understanding human drivers and predicting their behavior, and engaging in kinematic maneuvers (Barla, 2022). However, for this paper, focusing solely on environment sensing is sufficient.

The most common sensors used in real-life AV training include cameras, LiDAR, and RADAR (Barla, 2022). However, for the scope of this paper, ray casting will be utilized for environment sensing and obstacle detection. Utilizing real-life sensors introduces more complexity, while ray casting has already been implemented in Unreal Engine and is sufficient to give the agent the necessary information.

6.6 UNREAL ENGINE LEARNING AGENTS

Learning Agents (LA) is an Unreal Engine ML plugin for AI bots. It supports reinforcement and imitation learning to create game-playing agents, physics-based animations, automated QA bots, and more. It is not a general-purpose ML framework, so generative and conversational AI are not supported (Epic, 2024).

LA is intended for developers with different ML knowledge levels. Beginners can take advantage of the available plugins in Blueprints, while more advanced users can utilize the C++ API for greater control and flexibility (Epic, 2024; HuggingFace, n.d.).

The plugin comes with a built-in PyTorch PPO algorithm for RL. Functionality for defining observations, actions, and rewards is provided (Epic, 2024).

The Blueprint interface will be used to implement functionalities and demonstrate key concepts, enhancing the readability of this paper.

LA comes with a straightforward implementation of RL. The plugin is built around the LA Manager, where the rest of LA is constructed. It stores references to the various agents and is used to specify learning logic, such as batch size, number of iterations, etc. Listeners such as the *Interactor* and the *Trainer* are used to extend the existing functionality of the manager (see Appendix A: Learning Agents Framework for Blueprint examples) (Epic, 2024).

The interactor can be seen as the sensors and limbs, while the trainer is the coach. The interactor interprets the environment and takes actions, and the trainer determines if those actions are good or bad. The interactor accounts for the feedback received from the trainer when making decisions.

6.6.1 INTERACTOR

The interactor is utilized to specify the observations and actions of the agent. All agents share a common schema of observations and actions for a given manager. Four functions need to be overridden to complete the interactor (Epic, 2024).

SpecifyAgentObservation is used to specify all the observations. The manager calls this function once during setup. Numerous pre-defined observations come with the plugin (Epic, 2024).

After the observations are specified, they must be gathered. This step is carried out through the **GatherAgentObservation** function. For each observation specified, a corresponding gather node must be provided (Epic, 2024).

SpecifyAgentAction and **PerformAgentAction** are used to specify and perform actions. The functions are part of the Interactor (Epic, 2024).

6.6.2 TRAINER

The trainer, as the name suggests, is responsible for training the agents to perform the correct actions based on the observations they see. Two functions need to be implemented: rewards and completions (Epic, 2024).

GatherAgentReward rewards the actor for desired behavior and penalizes it for incorrect actions. As always, the plugin includes multiple pre-defined nodes.

When the agent gets into a bad state where no meaningful learning can occur, for example, driving off the track, we can end the training early. This speeds up training and can be specified in **GatherAgentCompletion** (Epic, 2024)**.**

There are two categories: terminations and truncations. Terminations are used when the agent is assumed to receive a zero reward for all further steps. Truncations are used when an early end of the episode is expected, but where the agent would have continued to receive more rewards (Epic, 2024).

7. RESEARCH

7.1 INTRODUCTION

This research aims to train autonomous vehicles (AVs) capable of navigating a road and avoiding static obstacles randomly placed along the path to determine how the policy batch size affects the obstacle avoidance success rate. The percentage of obstacles avoided will be gathered for each batch size to answer the research question. Five AVs will be trained using identical hyperparameters, the only variable being the policy batch size. After training, the AVs will be tested on three distinct track types: linear, circular, and zigzag. Data on obstacle avoidance performance will be collected from each track. The findings will be compared to draw meaningful conclusions. The detailed methodology is explained in the following sections.

7.2 EXPERIMENTAL SETUP

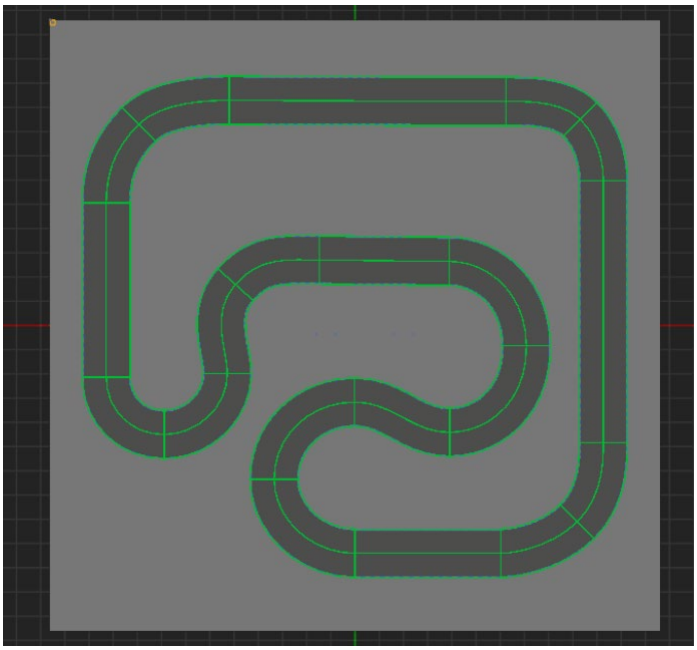
All the experiments in this paper are carried out in Unreal Engine (UE) 5.4. The engine offers a built-in vehicle template and a racing track map, significantly reducing development time. Another key reason for choosing UE is to explore and test the Learning Agents (LA) plugin, which is relatively new and underrepresented in academic research.

The Landscape tool in UE enables the easy creation of different track layouts, saving significant time when designing multiple track configurations. Additionally, UE provides powerful debugging tools that are particularly beneficial for RL, where debugging can often be challenging.

This project utilizes LA 5.4. Although it is an experimental plugin, it perfectly serves the purposes of this research.

The default track layout, as shown in Figure 4, is used for the training iterations. This layout represents a balanced middle ground, as it is neither too complex nor too simple. The track's length is 1017m.

Figure 4: Track layout used for training AVs



UE SportsCar_Pawn, with the vehicle template, is utilized for training and testing. No changes have been made to the vehicle set-up.

Figure 5: Vehicle used for training and testing

Five agents are trained over 5,000 iterations, each with a distinct batch size: 16, 32, 64, 128, and 256. Multiple instances of the same agent are trained simultaneously to accelerate the training process. The training is conducted in five phases, each focusing on one batch size and involving 16 agents training concurrently. Upon completion of each phase, the training is halted, and the training data is saved into four Unreal Data Assets: Critic, Decoder, Encoder, and Policy. After the training, every batch size is tested on three different track layouts to gather information about obstacle avoidance.

7.3 TRAINING

The training of a single agent is divided into two stages. In the first stage, the agent focuses on completing a full lap while staying within the track boundaries. Once this is achieved and the implementation is verified, static obstacles are introduced into the environment. In the second stage, the agent's behavior is updated to account for these obstacles, and the new goal becomes navigating the track while avoiding collisions with obstacles. This is achieved through an iterative research process. During this process, a policy batch size of 512 is used.

7.3.1 LEARNING TO DRIVE

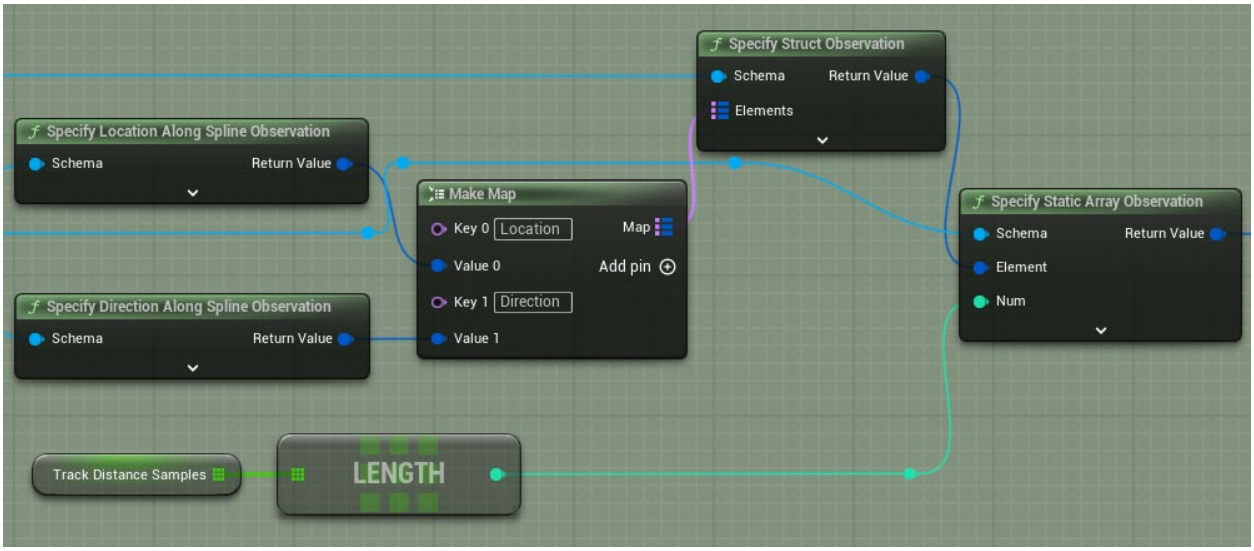
This section is fully implemented based on *Learning to Drive (5.4)* and *Improving Observations & Debugging (5.4)* tutorials provided by Epic Games (Epic, 2024).

As discussed in *Unreal Engine Learning Agents* Chapter, the Interactor and the Trainer, must be implemented.

SPECIFYING OBSERVATIONS

Multiple important observations must be made to make the agent follow the road and stay within bounds. The agent must be aware of the nearest point to the spline that follows the road, and it must know the direction of the spline at that point. To enhance the "vision" of the agent, the above-mentioned observations are done at different distances, allowing the agent to "look ahead" and be aware of the environmental changes that will occur. A *static array observation*, consisting of 7 struct observations, is utilized. Each item in the struct holds a pair of *Location* and *Direction* observations.

Figure 6: World observation of the agent

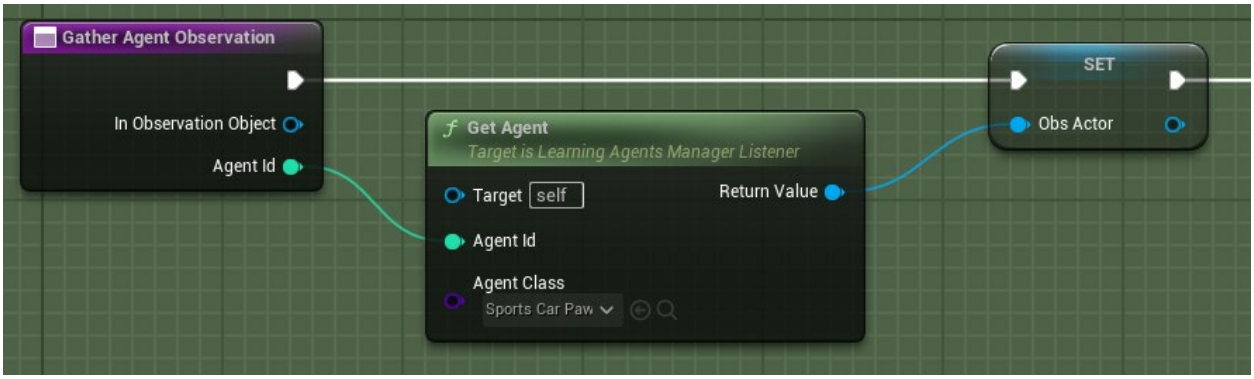


Another crucial observation is the agent's velocity, which will be used to reward the agent for driving as fast as possible (see Appendix B: Learning To Drive For Blueprint implementation).

GATHERING OBSERVATIONS

For this step, the agent must gather information such as its velocity. The agent can be accessed as follows:

Figure 7: Accessing agent in GatherAgentObservation in Learning Agents



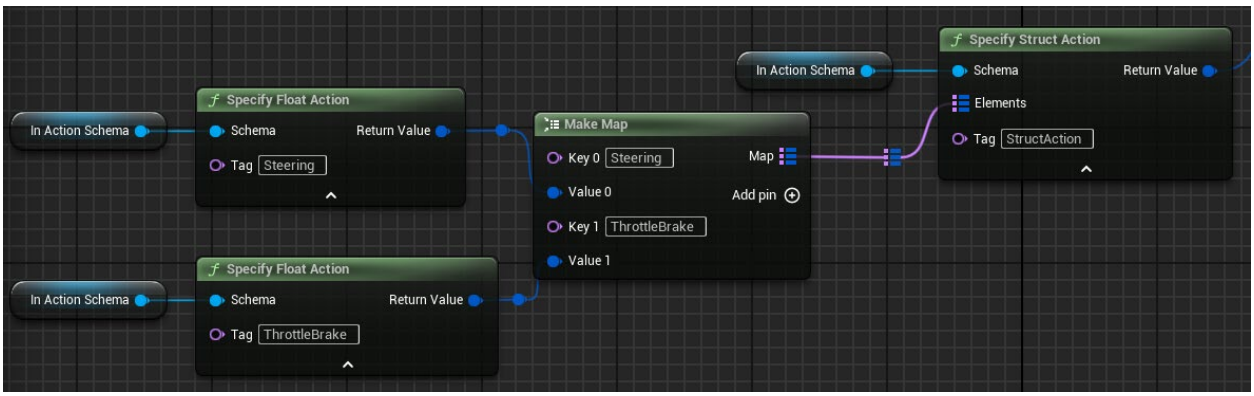
The data is gathered following the same structure specified in *SpecifyAgentObservation*. Each node used during the specification has a corresponding node for gathering the set information. The track spline is utilized to gather observations for the agent's vision. Using the *GetDistanceAlongSplineAtLocation* node, the sample distance is calculated for the following lookahead observations: 5m, 10m, 15m, 20m, 25m, and 30m. This data gathers direction and location observations along the track spline.

Gathering velocity observation data is straightforward. It is accessed directly using the agent (see Appendix B: Learning To Drive For Blueprint Implementation).

SPECIFYING ACTIONS

Two actions are necessary for the agent to navigate the world: throttle/brake and steering. A struct action is specified, consisting of two float actions (see Figure 8). Float actions are used because the *Vehicle Movement Component* in Unreal Engine expects a float value for steering, throttle, and brake inputs (see Appendix B: Learning To Drive For Blueprint implementation).

Figure 8: SpecifyAgentAction function implementation for autonomous driving

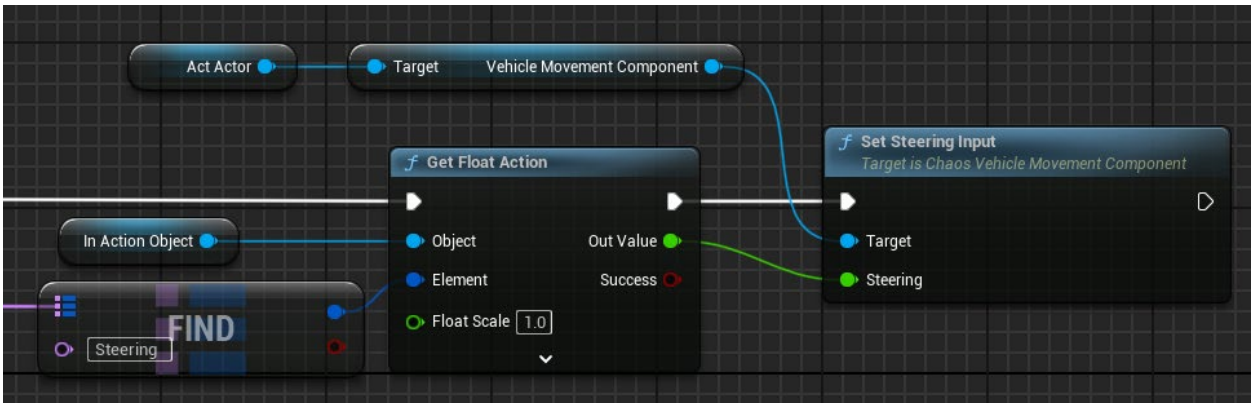


PERFORMING ACTIONS

Action-performing logic is provided to finalize the Interactor. The float value of *Steering Action* is fed into the *SetSteeringInput* function of the *Vehicle Movement Component*, enabling the agent to steer on the road. The value is between -1 and 1 (see Figure 9).

Throttle/brake logic is more complex. If the float action yields a positive number, that number is fed into the *SetThrottleInput* function, and *SetBrakeInput* takes 0. If the resulting number is negative, then the positive of that number is used as an argument in *SetBrakeInput*, while *SetThrottleInput* takes 0 (see Appendix B: Learning To Drive For Blueprint implementation).

Figure 9: Perform steering logic



GATHERING REWARDS

The agent is rewarded for driving as fast as possible along the spline. This is insufficient because the agent still leaves the track bounds, considering it is not penalized. A negative reward is introduced if the distance between the agent and the spline is greater than the distance between the spline and the road “walls” (see Appendix B: Learning To Drive For Blueprint implementation).

RESULT

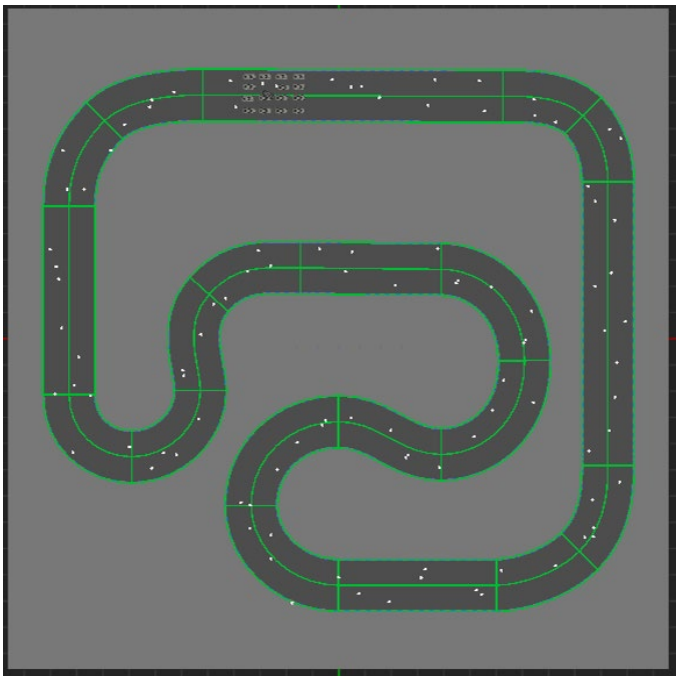
Using the above-mentioned configuration, the agents learn to drive around the training track (see Figure 4) without leaving its bounds. After training 16 agents simultaneously, the driving skills are significantly improved if the simulation runs for at least 30 minutes. After two hours of training, the agent can navigate the track infinitely without leaving its bounds.

Figure 10: Autonomous driving agent after two hours of training

7.3.2 OBSTACLE AVOIDANCE

Static obstacles are randomly distributed across the track to introduce obstacle avoidance into the training. Every time the simulation is run, a change in obstacle layout is guaranteed (See Figure 11).

Figure 11: Racing track populated with static obstacles



The obstacle is an *Actor Blueprint* containing a cube mesh and a box collider. Those actors are created using the *SpawnActor* node and are guaranteed to be within the bounds of the track (see Appendix C: Obstacle Avoidance – Setup For Blueprint implementation).

The interactor and the trainer must be updated to account for obstacles across the track. There are multiple ways to achieve the desired behavior, so different approaches will be explored until one that suits the needs of this paper is found. When a set approach does not improve after two hours of training, it is assumed unsuitable for this research.

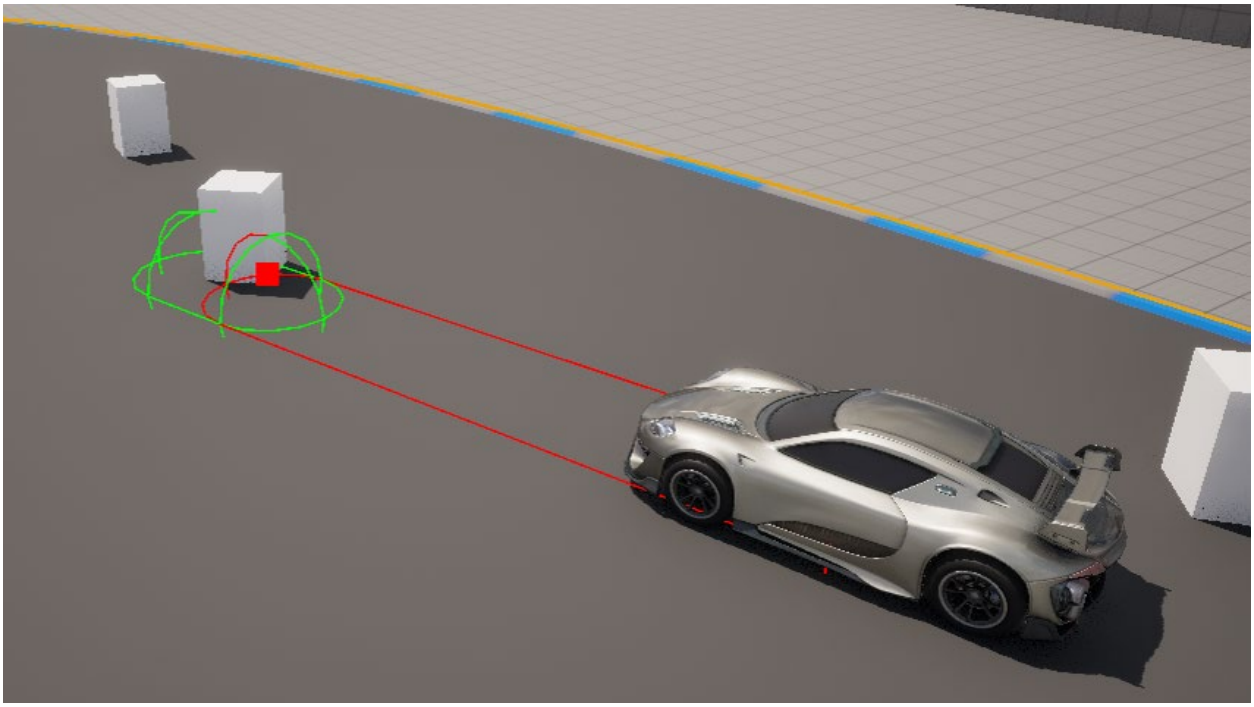
APPROACH 1: SINGLE SPHERE TRACE

The most intuitive approach is giving “eyes” to the agent (see Figure 12). A singular sphere trace is utilized to progress naturally toward a more complex solution. From the agent’s location, a sphere with a radius of 1m is swept along the forward vector of the agent. Objects that collide with the sphere up to 10m ahead are detected. The location of the hit is registered and observed by the agent. A penalty of -10 is given if the angle between the front vector and the vector from the agent to the hit location is less than 36.87°. This number is the result of an iterative process (see Appendix D: Obstacle Avoidance – Approach 1 For Blueprint Implementation).

After two hours of training, the desired behavior is not achieved. The agent consistently hits most of the obstacles present on the road. After careful investigation, two significant problems appeared with this implementation. One of them is the representation of the vision. One sphere cast is not enough, especially when the agent is taking a turn; a collision can happen without the vision detecting the obstacle. Another problem is the penalty design. A set angle difference of 36.87° proves unreliable, even if an obstacle is detected on time. Due to time constraints, this

study does not investigate other angles. For this reason, different penalty designs are introduced in the following approaches.

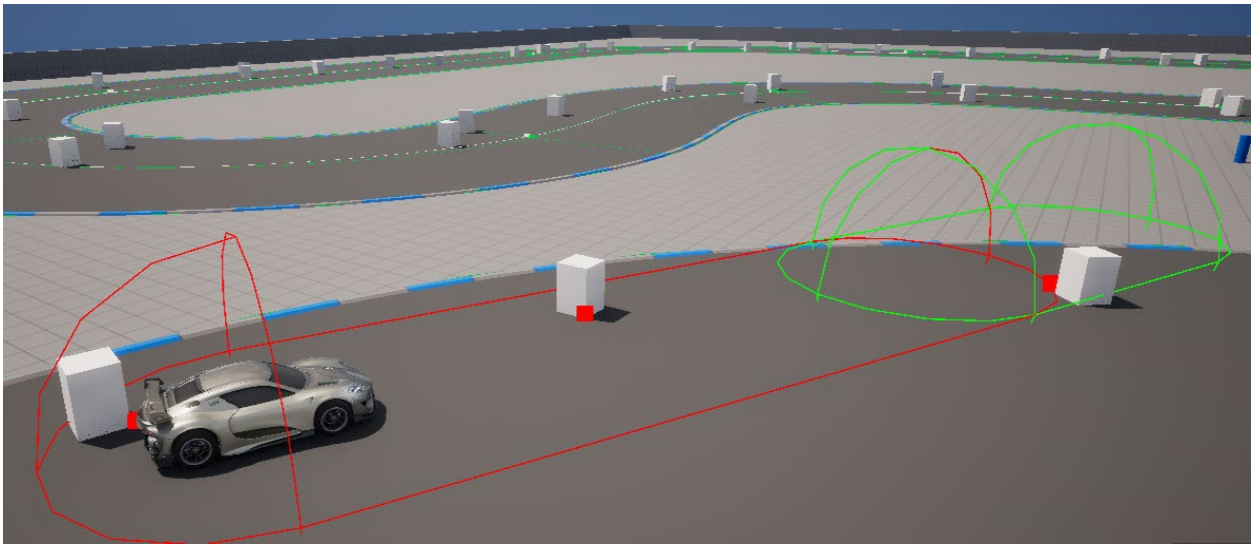
Figure 12: Agent with a singular sphere cast detecting an obstacle



APPROACH 2: SINGLE SPHERE TRACE WITH MULTIPLE OBSTACLE OBSERVATION

To enhance the functionality of the existing obstacle avoidance, not one but all obstacles are observed. The sphere cast radius is increased to 4m, encapsulating a much larger area than the previous approach. The vision still consists of a singular “eye”. A penalty of -10 is given if the agent collides with an obstacle (See Appendix E: Obstacle Avoidance – Approach 2 For Blueprint implementation).

Figure 13: Agent with a singular sphere cast and enhanced radius



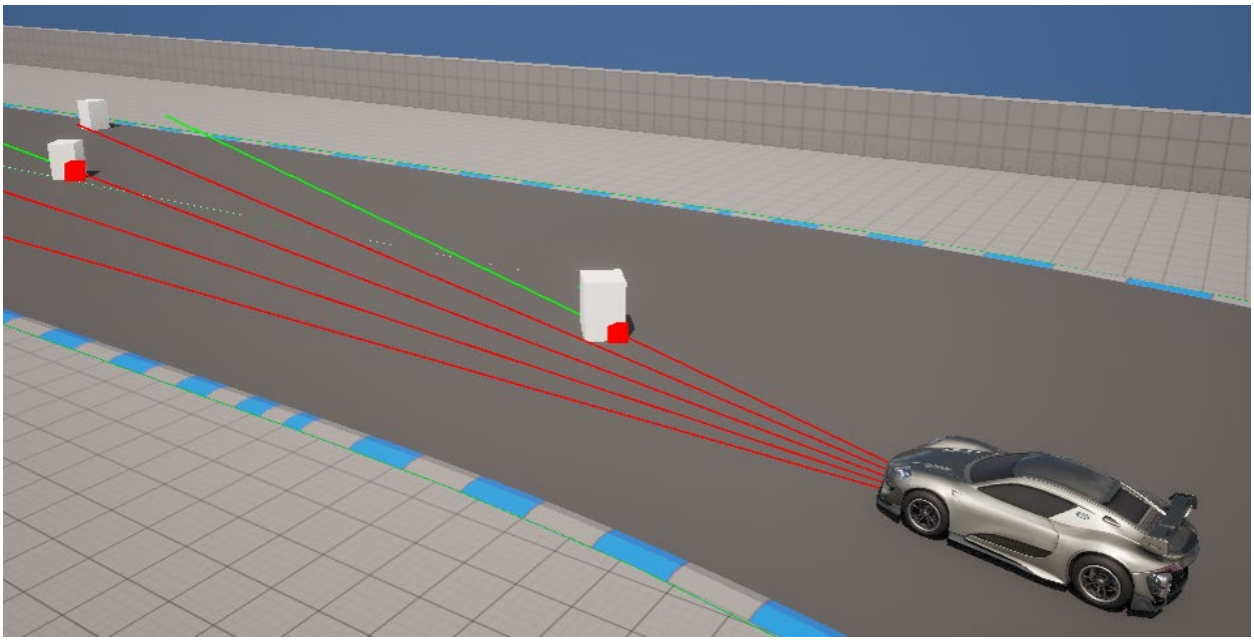
After two hours of training, the observable behavior is slightly improved. This is because the agent's area of vision is much larger than previously. However, the behavior is still unreliable. The agent consistently hits the obstacles.

A major conclusion from the experiments is that the agent must observe the surroundings as much as possible, anticipate the collision, and be punished before the collision occurs.

APPROACH 3: MULTIPLE LINE TRACES

The agent's vision is extended to account for the shortcomings of the previous approaches. Five line traces are cast, each with a relative angle to the forward vector. The angles are -10°, -5°, 0°, 5° and 10°. The distance of the vision is still 10m. Obstacles detected by the agent's vision are saved in an array. A new observation of type *bool* is made. The agent observes if this array of detected obstacles is empty or not. This information is used to give a penalty of -10 if this array is not empty, meaning that the agent gets a penalty every time it detects an obstacle (see Appendix F: Obstacle Avoidance – Approach 3 For Blueprint implementation). This approach is pursued out of curiosity to observe how the agent will behave when detecting an obstacle, which is considered a bad action. The assumption is that the agent will learn to "look away" from any obstacle once detected, leading to naturally avoiding it.

Figure 14: Agent with five lines representing the vision

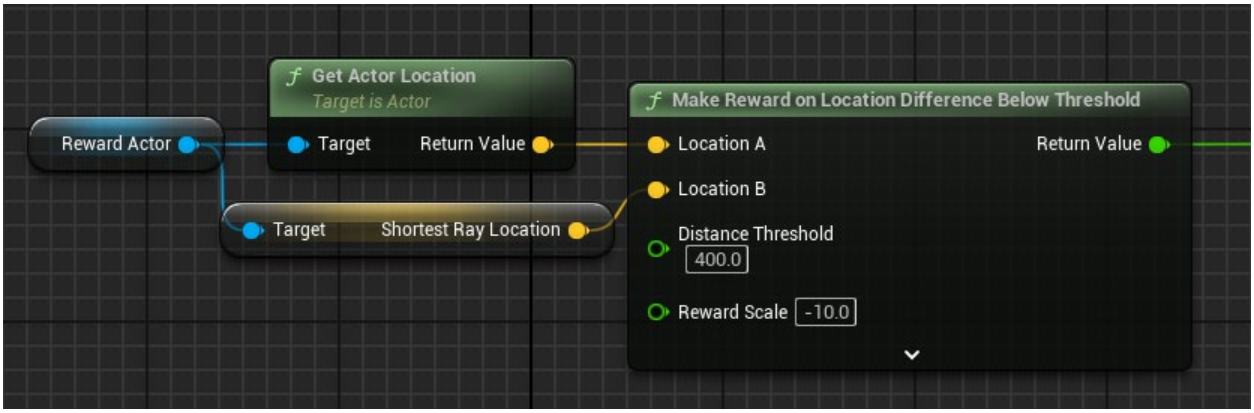


After two hours of training, the agent's behavior has changed noticeably. As anticipated, the agent tries to look away, causing it to navigate past the obstacles. The downside is that if an obstacle is too close to the side of the road, the agent drives off the road to avoid a collision. While this is an improvement, it still requires refining.

There are two places to look for improvements: interactor and trainer. An upgrade to the interactor would mean adding additional lines to represent the vision. In contrast, an upgrade to the trainer would indicate either experimenting with different reward values or designing a new reward system. To detect where the error lies, the interactor will stay untouched. A new reward will be introduced, replacing the existing negative reward for collision.

When the agent gets closer than 40m to the closest obstacle detected, it is given a reward of -10.

Figure 15: Penalty for getting too close to an obstacle



This change demonstrates the importance of a good reward system design. After the following change, the behavior is notably improved. The agent tries to stay within bounds while avoiding obstacles. There are still situations where the agent collides with obstacles. Approach 4 solves the above issues by introducing the first working version of the agent, which is ready for training with multiple batch sizes.

APPROACH 4: SMART AGENT

The first change contributing to the desired outcome concerns the observations. To ensure the agent does not miss any obstacles ahead, two more rays are added to the vision, making it a total of seven rays between -45° and 45°. The vision range is also greatly enhanced. Now, the agent can see 100m ahead.

The reward system is also adjusted. If the agent gets closer than 50m to an obstacle within -11.7° and 11.7° relative to its forward vector, a significantly greater negative reward of -25 is given. To further enhance the reward system, upon collision, a negative reward of -50 is given (see Appendix G: Obstacle Avoidance – Approach 4 For Blueprint implementation).

The above-mentioned changes result in a competent agent avoiding obstacles after two hours of training. As a note, the values and approaches used that result in a smart agent are part of an iterative process. They do not represent the best or most optimal results but are sufficient for this paper. For this reason, the agent's

configuration will not change from now on, and training iterations for each batch size, as provided before, will be conducted.

Figure 16: Multiple agents driving around the track

7.3.3 RESULTS

With the implementation finalized, multiple training phases have been conducted to prepare all agents for testing. Each batch size, namely 16, 32, 64, 128, and 256, was trained separately, utilizing 16 parallel instances to accelerate the process. The training was halted after 5,000 iterations.

See Appendix H: Training Results for a detailed explanation of the training results.

7.4 TESTING

7.4.1 SETUP

Testing episodes are carried out on different tracks. Each batch size is tested for 50 episodes on a linear, circular, and zigzag track, represented by a singular agent. The obstacle layout stays consistent for each training, making batch size the only variable. Each track contains 35 obstacles. Two rays, originating from the agent's location and facing left and right, are utilized to keep track of avoided obstacles. When the rays collide with an obstacle, meaning the agent is next to it, it is considered avoided. A testing episode is halted if the agent:

- Collides with an obstacle
- Drives off the road
- Finishes a full lap, indicating a 100% obstacle avoidance rate

After each testing episode, the number of obstacles avoided during that episode is saved. After completing 50 episodes, the results are written in a CSV file.

Figure 17: Zigzag, circular, and linear tracks used for testing

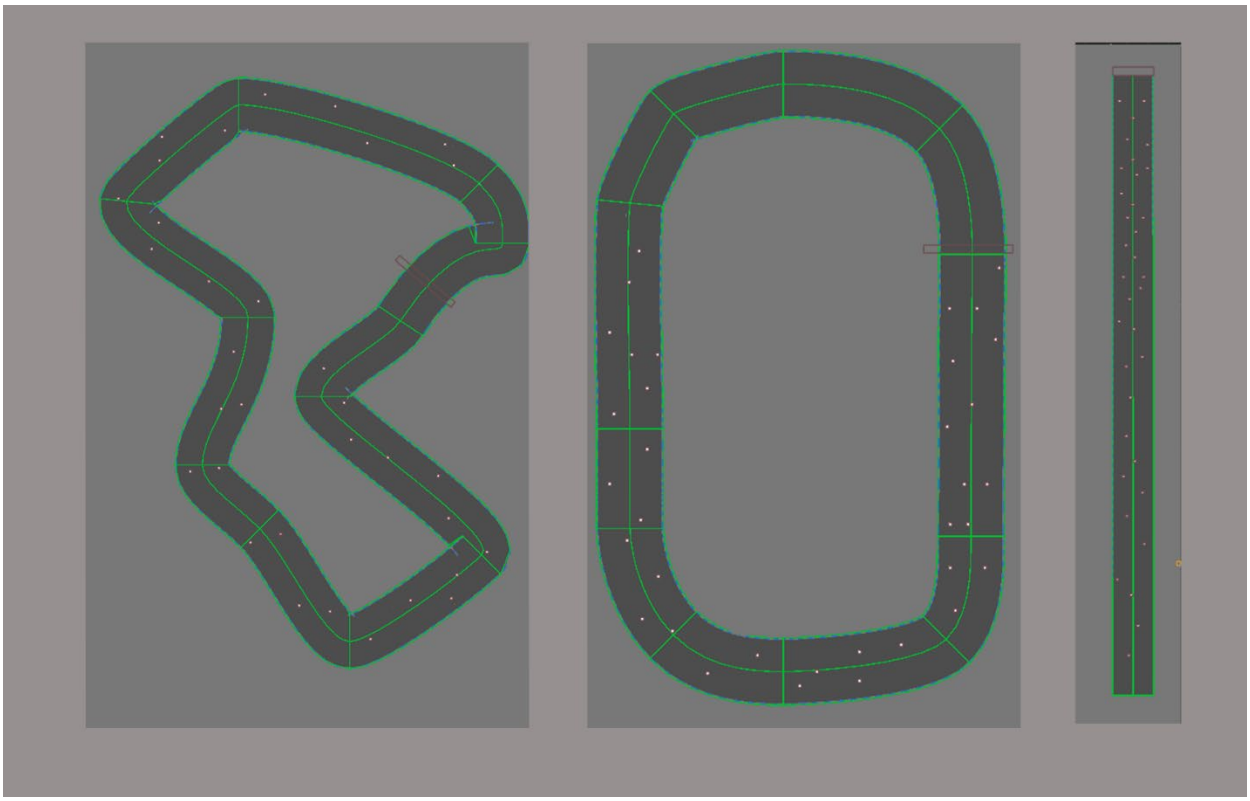
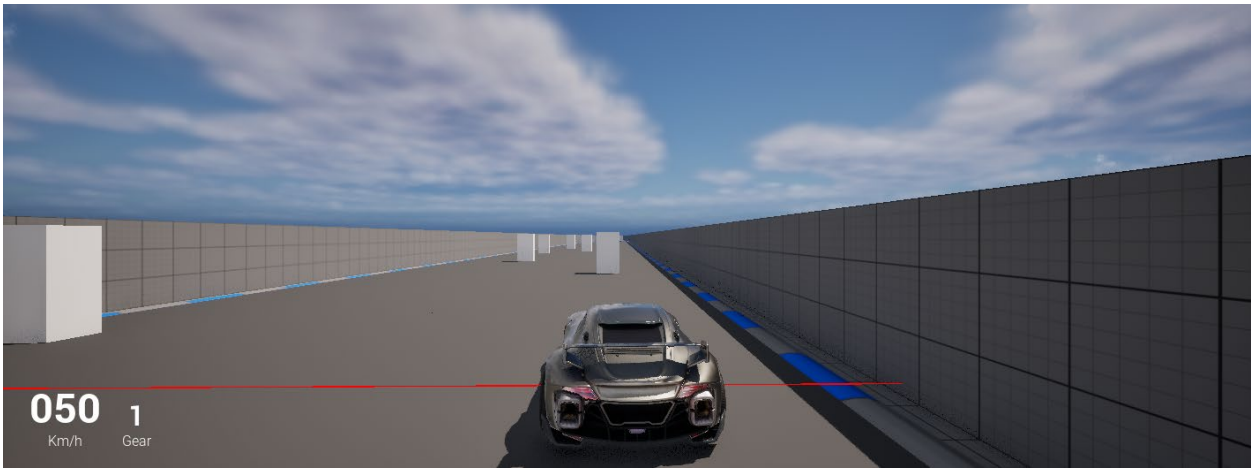


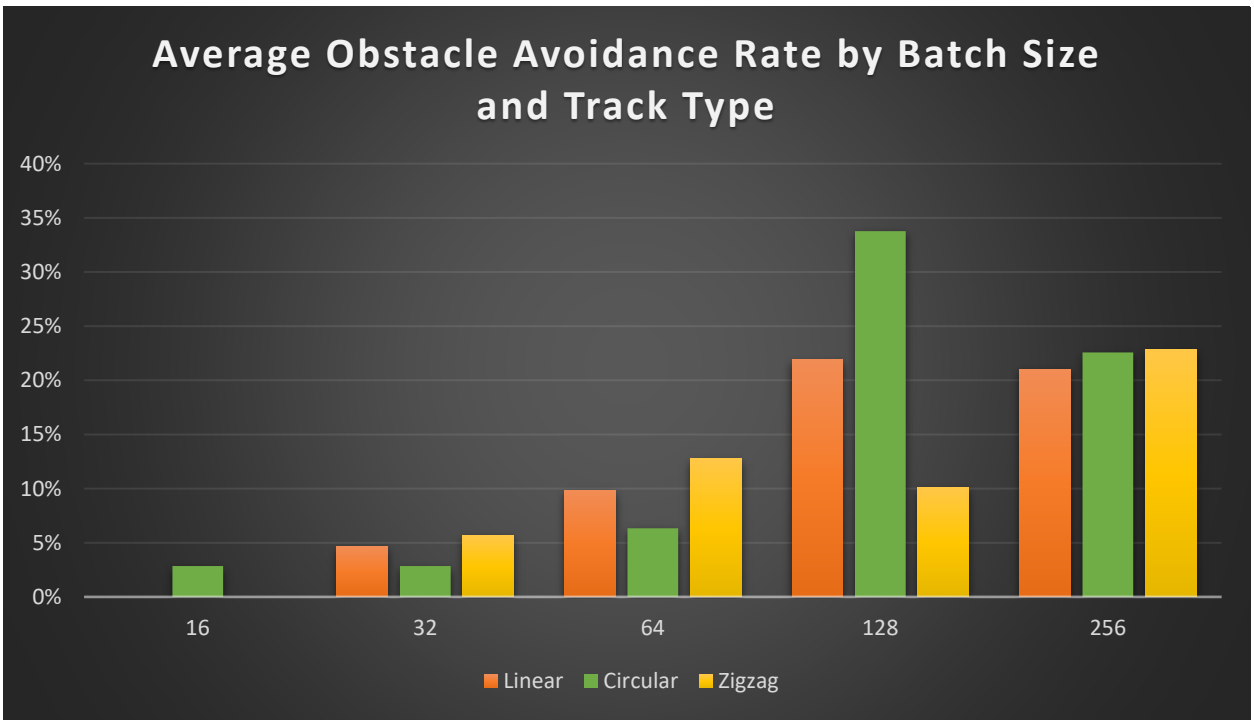
Figure 18: Agent running a test episode on a linear track



7.4.2 RESULTS

Figure 21 Shows the average obstacle avoidance rate across three track types for different batch sizes. Each bar represents the performance for a specific track type at a given batch size. The vertical axis represents the percentage of obstacles avoided. The horizontal axis represents the different batch sizes.

Figure 19: Chart showing the average obstacle avoidance rate across three track types for different batch sizes



Batch size 16 performs very poorly across all track types. Batch size 32 slightly improves compared to 16, but the performance stays under 10%. Batch 64 outperforms the smaller batch sizes, but the success rate stays below 15%. Batch 128 achieves the highest obstacle avoidance rate on the circular track, almost reaching 35% while

underperforming on the zigzag track. Batch 256 delivers the most consistent results, with obstacle avoidance rates clustering between 20% and 25%.

8. DISCUSSION

The results, as shown in Figure 21, show a clear relationship between batch size and average obstacle avoidance rate across all track types. Small batch sizes (16 and 32) indicate unstable training because the obstacle avoidance rates are close to 0%. This shows that a smaller batch size does not guarantee global optima, which confirms the statement by (Shen, 2018), that small batch sizes have faster convergence to “good” solutions. Performance slightly improves as the batch size increases to 64, but the results remain suboptimal.

The performance peaks at a batch size of 128, with the circular track's highest obstacle avoidance rate (33.77%). This suggests that this batch size best balances training stability and model generalization. However, on the zigzag track, the success rate remains relatively low (10.11%). This shows that the model struggles with the added complexity of sharp turns and irregular patterns. This suggests that while batch size 128 optimizes performance for more straightforward track types, it may not fully address the challenges posed by more complex environments like the zigzag track.

At a batch size 256, performance plateaus across all track types, with minor improvements. This is likely due to the reduced generalization capacity associated with overly large batch sizes. It is safe to say that bigger batch sizes performed much better than smaller ones.

The overall low percentages highlight a significant limitation in the model’s ability to effectively learn and apply obstacle avoidance strategies, regardless of batch size. This could come from insufficient training episodes, suboptimal hyperparameters, limitations in the PPO algorithm itself, or the implementation methods of obstacle avoidance.

The results provide insights into the influence of batch size, but the overall low performance suggests a broader challenge in training reinforcement learning models for obstacle avoidance.

Factors such as the implementation of training techniques, the design of the reward system, and the overall architecture of the learning environment likely contribute far more significantly to the obstacle avoidance success rate.

9. CONCLUSION

This research investigated how policy replay batch size influences the static obstacle avoidance success rate for an autonomous vehicle trained using Proximal Policy Optimization within Unreal Engine's Learning Agents framework. The findings partially support the hypotheses while highlighting broader challenges in achieving practical obstacle avoidance.

Hypothesis 1, which posited that there exists an optimal batch size greater than 64 that maximizes the success rate, was partially supported by the results. Batch sizes more significant than 64 performed better overall, but they challenge the idea of a universally optimal batch size. The results point to a context-dependent relationship where the optimal batch size varies based on task complexity and the ability of the model to generalize across different scenarios.

Hypothesis 2, on the other hand, proposed that larger batch sizes (128 and 256) would outperform smaller ones, and this was validated. Smaller batch sizes exhibited poor performance, with very low success rates for 16 and 32, highlighting unstable training. However, the plateau in performance between 128 and 256 challenges the assumption that larger batch sizes consistently lead to better outcomes.

To conclude this study, it is important to identify an optimal batch size, but focusing solely on this will not improve the obstacle avoidance drastically. A more comprehensive approach is needed to overcome the challenges of reinforcement learning in dynamic environments.

10. FUTURE WORK

This paper explored one approach to developing autonomous driving using Reinforcement Learning. In machine learning, parameter tuning plays a critical role, so future work could focus on optimizing the balance between rewards and penalties to enhance obstacle avoidance further. Another potential research idea could involve dynamic obstacle avoidance and analyzing how batch size influences the performance in that scenario. Adding more observations and actions to the agent by incorporating traffic conditions could also provide a better understanding of autonomous driving. As the Learning Agents plugin evolves, new features and possibilities will likely emerge, opening further avenues for exploration and innovation in autonomous driving research.

11. CRITICAL REFLECTION

I began this research with no prior knowledge of Machine Learning. Through extensive reading and practical experimentation, I have developed a strong understanding of Reinforcement Learning, mainly how it can be applied to train agents for use in games. Even though my experiments focused on autonomous driving, I have realized that Reinforcement Learning principles apply across different domains.

Reflecting on the process, I would start the research process earlier. The development required significant time because testing the implementation required observing the agent's behavior for at least an hour before it could be confirmed or invalidated. I would also put less emphasis on tweaking values to improve the behavior and investigate implementation errors first. If I had more time, I would focus on fine-tuning my implementation, as the results indicated that my approach could have been improved.

This research has inspired me to explore Learning Agents further and experiment with their capabilities beyond this project's scope. I plan to follow the plugin's development closely.

Looking ahead, I am excited about the potential for Reinforcement Learning to become more integrated into everyday applications. This research has been both challenging and rewarding, providing me with valuable insights and a strong foundation upon which to build future projects.

12. REFERENCES

Arabaci, B. (2023, May 31). Observations/States Space. *Medium*. https://bahadirarabaci.medium.com/observations-states-space-c6a01f9b75ce

Bacancy. (2024). *Understanding Batch Size in Deep Reinforcement Learning*. https://www.bacancytechnology.com/qanda/qa-automation/batch-size-in-background-of-deep-reinforcement-learning

Barla, N. (2022, July 22). *Self-Driving Cars With Convolutional Neural Networks (CNN)*. Neptune.Ai. https://neptune.ai/blog/self-driving-cars-with-convolutional-neural-networks-cnn

Bi, Q., Goodman, K. E., Kaminsky, J., & Lessler, J. (2019). What is Machine Learning? A Primer for the Epidemiologist. *American Journal of Epidemiology*, *188*(12), 2222–2239. https://doi.org/10.1093/aje/kwz189

CFSS. (n.d.). *Autonomous Vehicles Factsheet | Center for Sustainable Systems*. Retrieved December 30, 2024, from https://css.umich.edu/publications/factsheets/mobility/autonomous-vehicles-factsheet

Crespo, J., & Wichert, A. (2020). Reinforcement learning applied to games. *SN Applied Sciences*, *2*(5), 824. https://doi.org/10.1007/s42452-020-2560-3

Epic, G. (2024, April 16). *Learning Agents (5.4) | Course*. Epic Games Developer. https://dev.epicgames.com/community/learning/courses/kRm/unreal-engine-learning-agents-5-4

GeeksforGeeks. (2024a). *How does reward maximization work in reinforcement learning?* GeeksforGeeks. https://www.geeksforgeeks.org/how-does-reward-maximization-work-in-reinforcement-learning/

GeeksforGeeks. (2024b). *How to Make a Reward Function in Reinforcement Learning?* GeeksforGeeks. https://www.geeksforgeeks.org/how-to-make-a-reward-function-in-reinforcement-learning/

HuggingFace. (n.d.). *An Introduction to Unreal Learning Agents—Hugging Face Deep RL Course*. Retrieved December 30, 2024, from https://huggingface.co/learn/deep-rl-course/en/unitbonus3/learning-agents

Jain, anushka. (2023). *Types of Machine Learning*. GeeksforGeeks. https://www.geeksforgeeks.org/types-of-machine-learning/

Janiesch, C., Zschech, P., & Heinrich, K. (2021). Machine learning and deep learning. *Electronic Markets*, *31*(3), 685–695. https://doi.org/10.1007/s12525-021-00475-2

jcm69. (2017, July 5). *Answer to "What is the advantage of keeping batch size a power of 2?"* [Online post]. Data Science Stack Exchange. https://datascience.stackexchange.com/a/20193

Kiran, B. R., Sobh, I., Talpaert, V., Mannion, P., Sallab, A. A. A., Yogamani, S., & Pérez, P. (2021). *Deep Reinforcement Learning for Autonomous Driving: A Survey* (No. arXiv:2002.00444). arXiv. http://arxiv.org/abs/2002.00444

Kumar. (2024, February 21). PPO Algorithm. *Medium*. https://medium.com/@danushidk507/ppo-algorithm-3b33195de14a

OpenAI. (2017). *Proximal Policy Optimization*. https://openai.com/index/openai-baselines-ppo/

OpenAI. (2018). *Part 1: Key Concepts in RL — Spinning Up documentation*. https://spinningup.openai.com/en/latest/spinningup/rl_intro.html

Shen, K. (2018, June 19). Effect of batch size on training dynamics. *Geek Culture*. https://medium.com/mini-distill/effect-of-batch-size-on-training-dynamics-21c14f7a716e

SmartLab, A. (2019, February 18). Reinforcement Learning algorithms—An intuitive overview. *Medium*. https://smartlabai.medium.com/reinforcement-learning-algorithms-an-intuitive-overview-904e2dff5bbc

Wen, M., Park, J., & Cho, K. (2020). A scenario generation pipeline for autonomous vehicle simulators. *Human-Centric Computing and Information Sciences*, *10*(1), 24. https://doi.org/10.1186/s13673-020-00231-z

13. ACKNOWLEDGEMENTS

I want to thank Alexandros Kougentakos for his invaluable assistance in troubleshooting my obstacle avoidance system. I am also profoundly grateful to Kasper Geeroms, my coach, for his support and to Fries Boury, my supervisor, for his continuous guidance throughout this project.

14. APPENDICES

14.1 APPENDIX LIST

- Appendix A: Learning Agents Framework
- Appendix B: Learning To Drive
- Appendix C: Obstacle Avoidance – Setup
- Appendix D: Obstacle Avoidance – Approach 1
- Appendix E: Obstacle Avoidance – Approach 2
- Appendix F: Obstacle Avoidance – Approach 3
- Appendix G: Obstacle Avoidance – Approach 4
- Appendix H: Training Results

14.2 APPENDIX A: LEARNING AGENTS FRAMEWORK

Figure A1: SpecifyAgentObservation function in Learning Agents 5.4

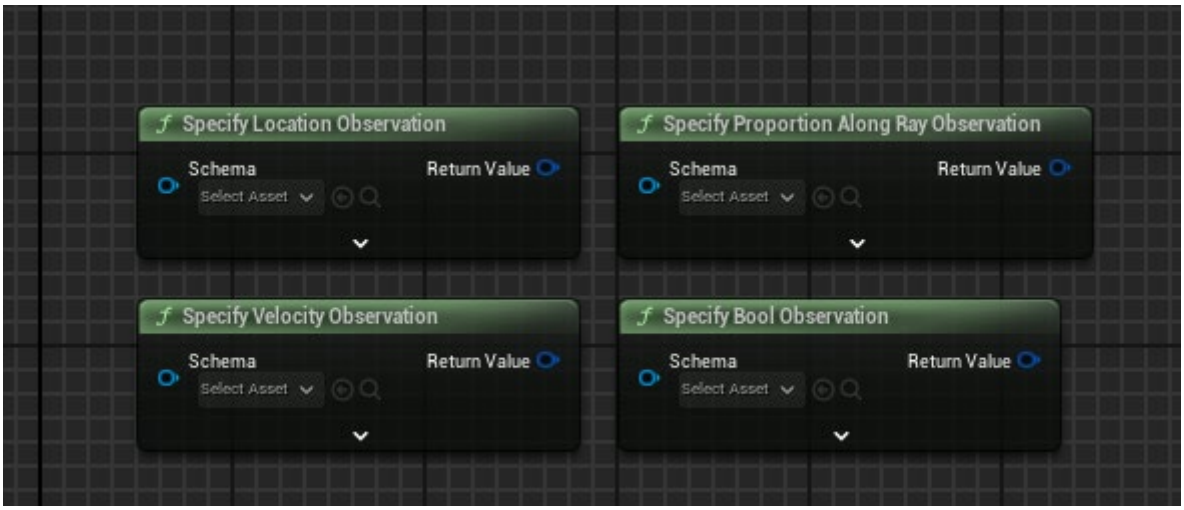


Figure A2: GatherAgentObservation Function in Learning Agents 5.4

Figure A3: SpecifyAgentAction and PerformAgentAction in Learning Agents 5.4

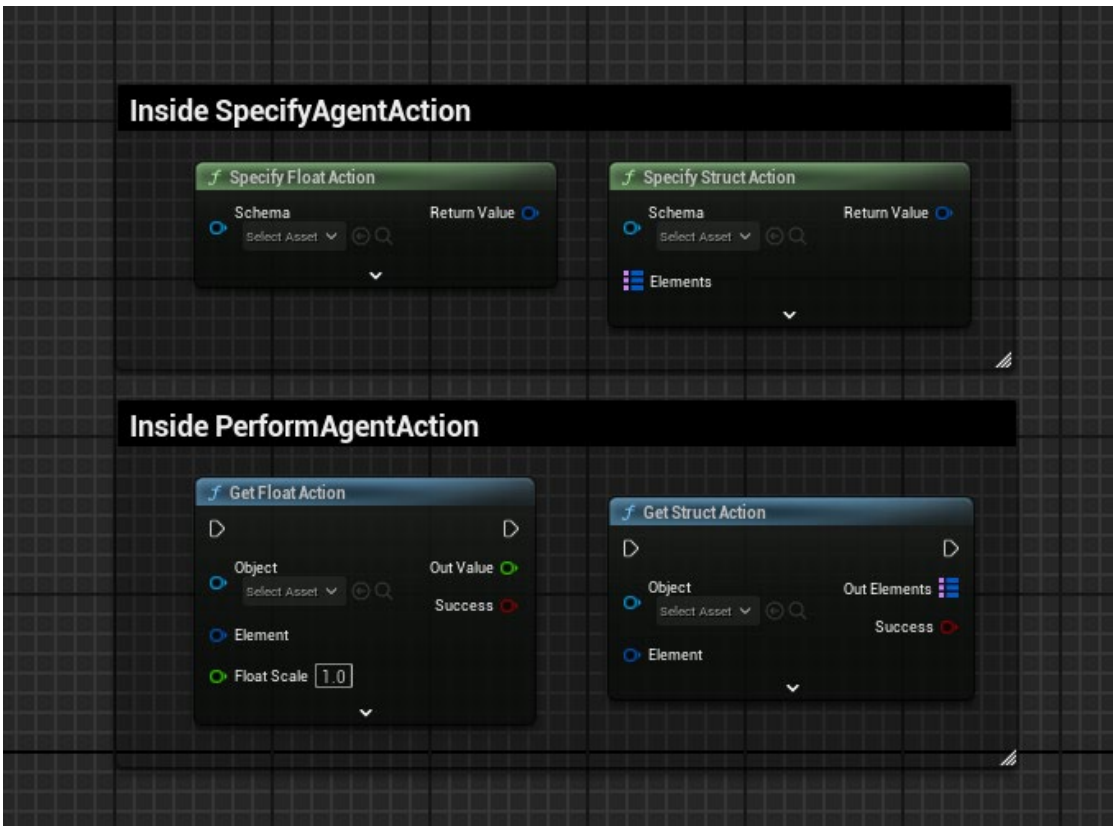
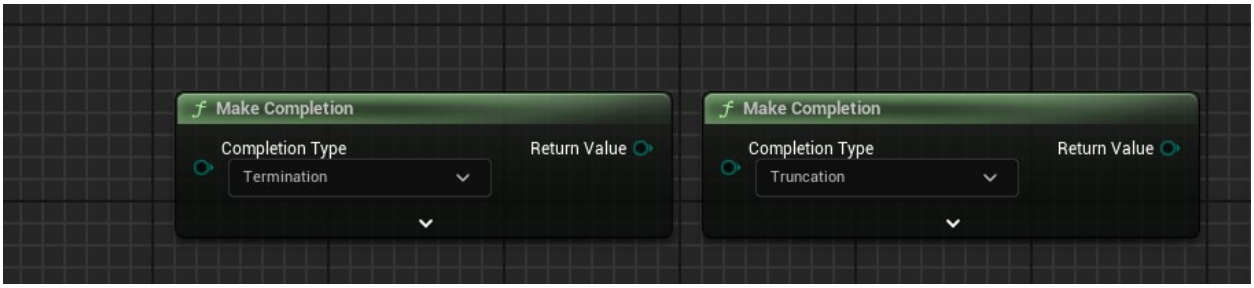


Figure A4: GatherAgentReward in Learning Agents 5.4

Figure A 5: GatherAgentCompletion in Learning Agents 5.4



14.3 APPENDIX B: LEARNING TO DRIVE

Figure B1: Velocity observation in Blueprints

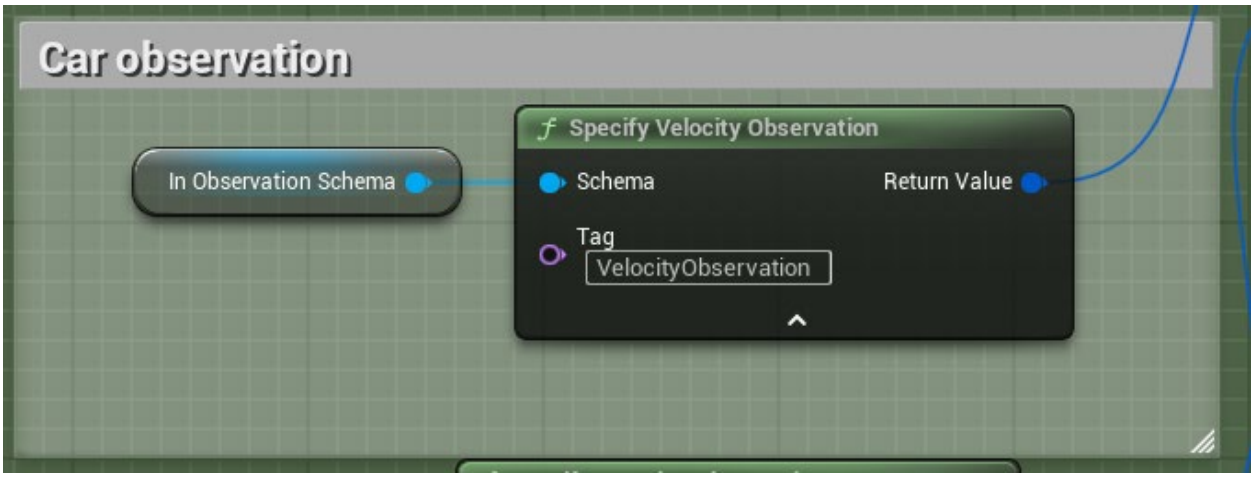


Figure B2: Mapping the observations together

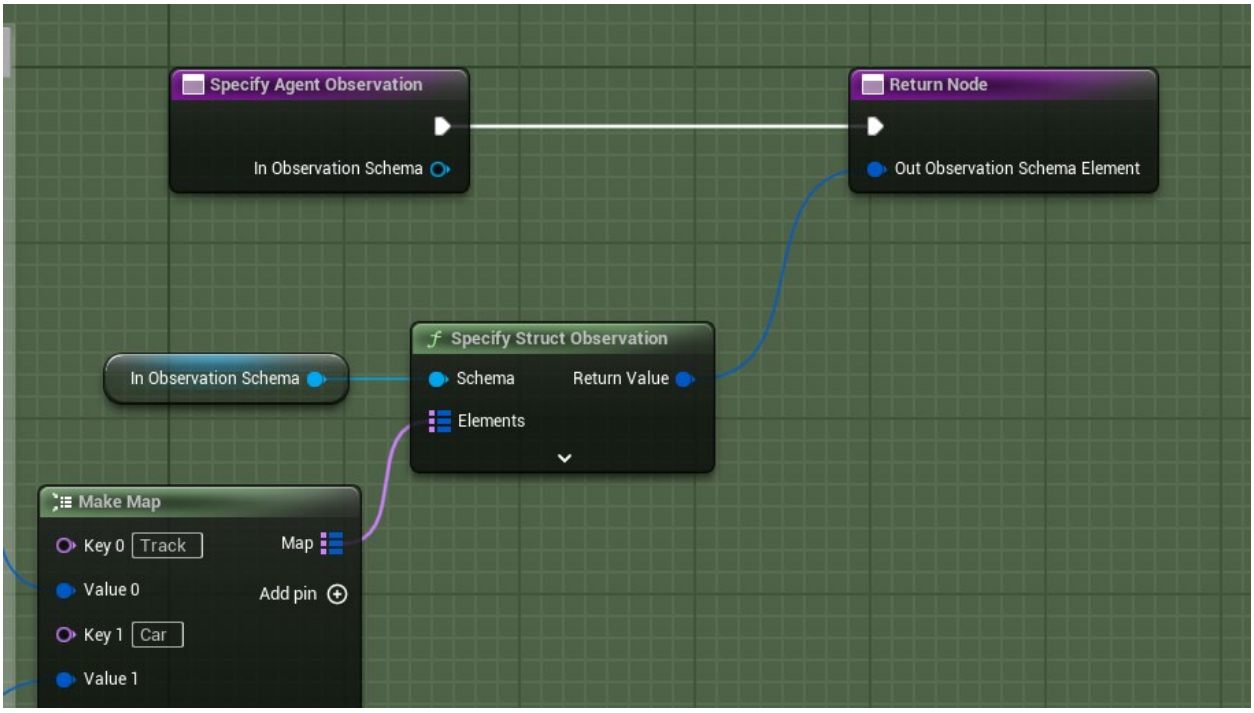


Figure B3: Track Distance Samples Array

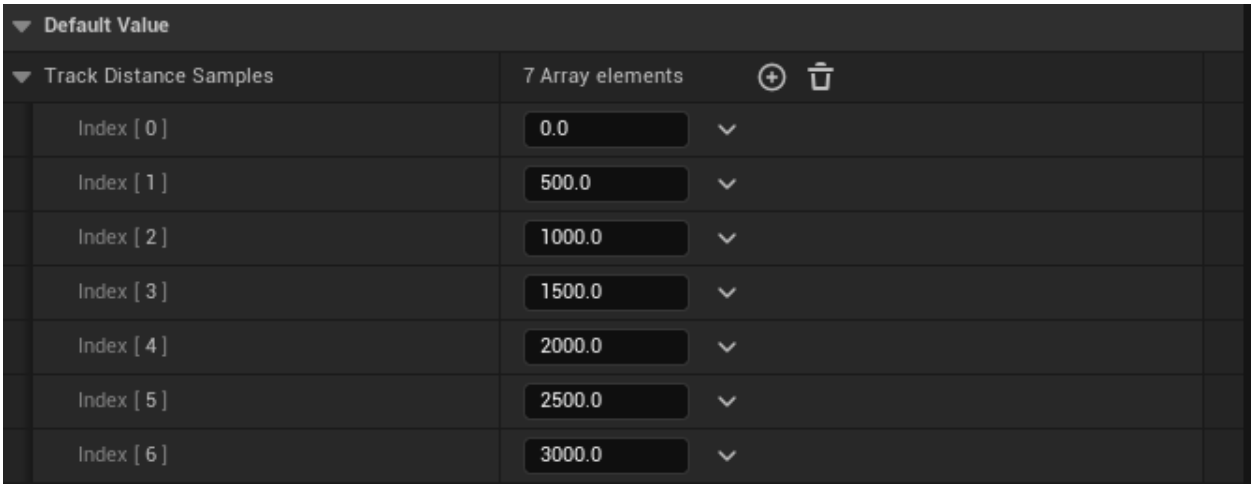


Figure B4: Finding Sample Distance Per Sample

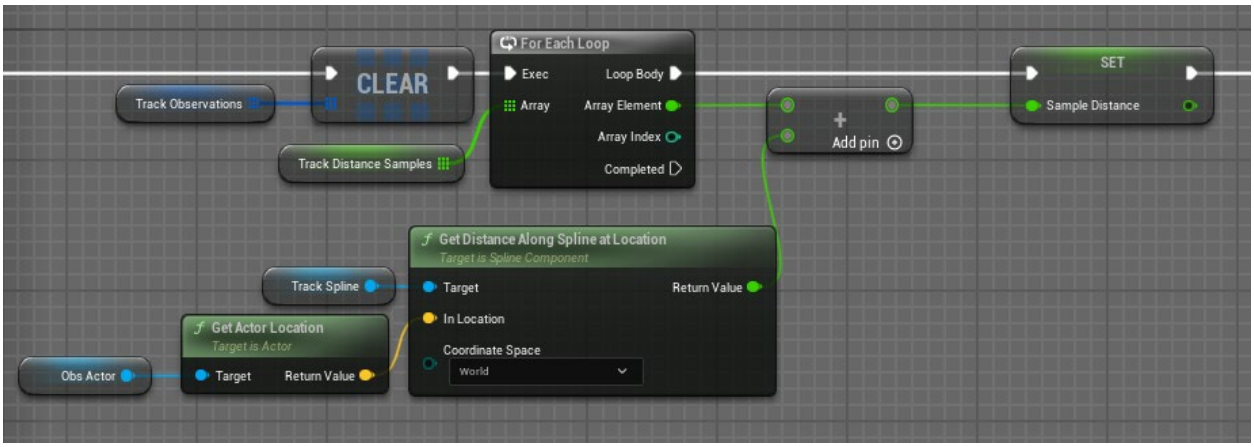


Figure B5: Gathering Location & Direction Along Spline Observations

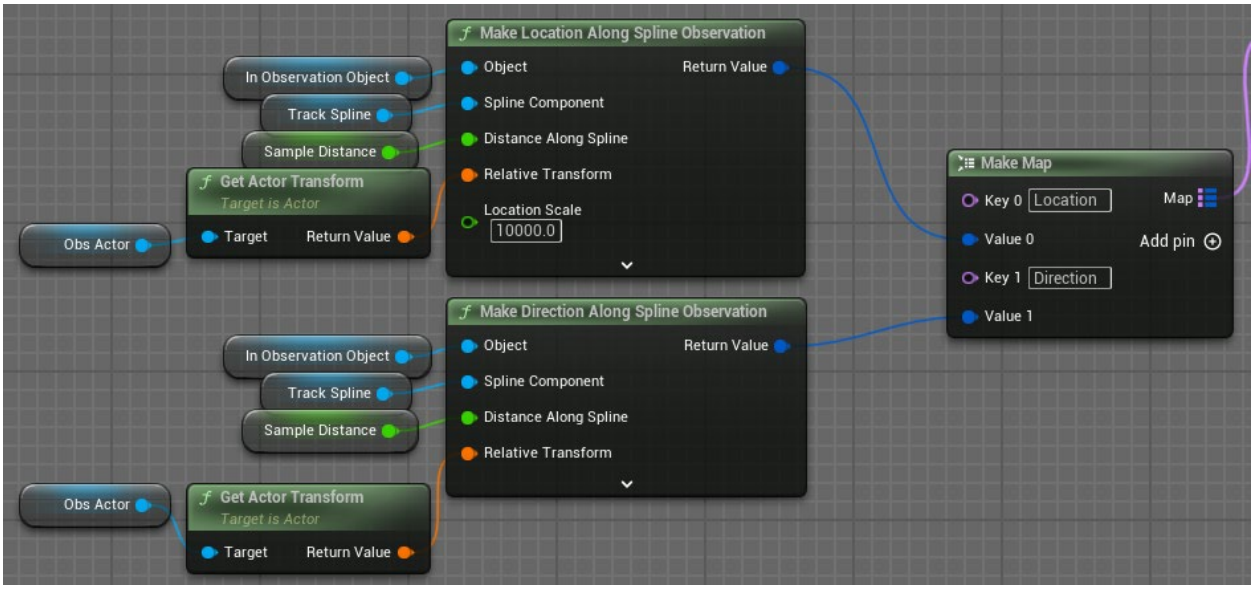


Figure B6: Saving Location and direction Along Spline Observations into an array

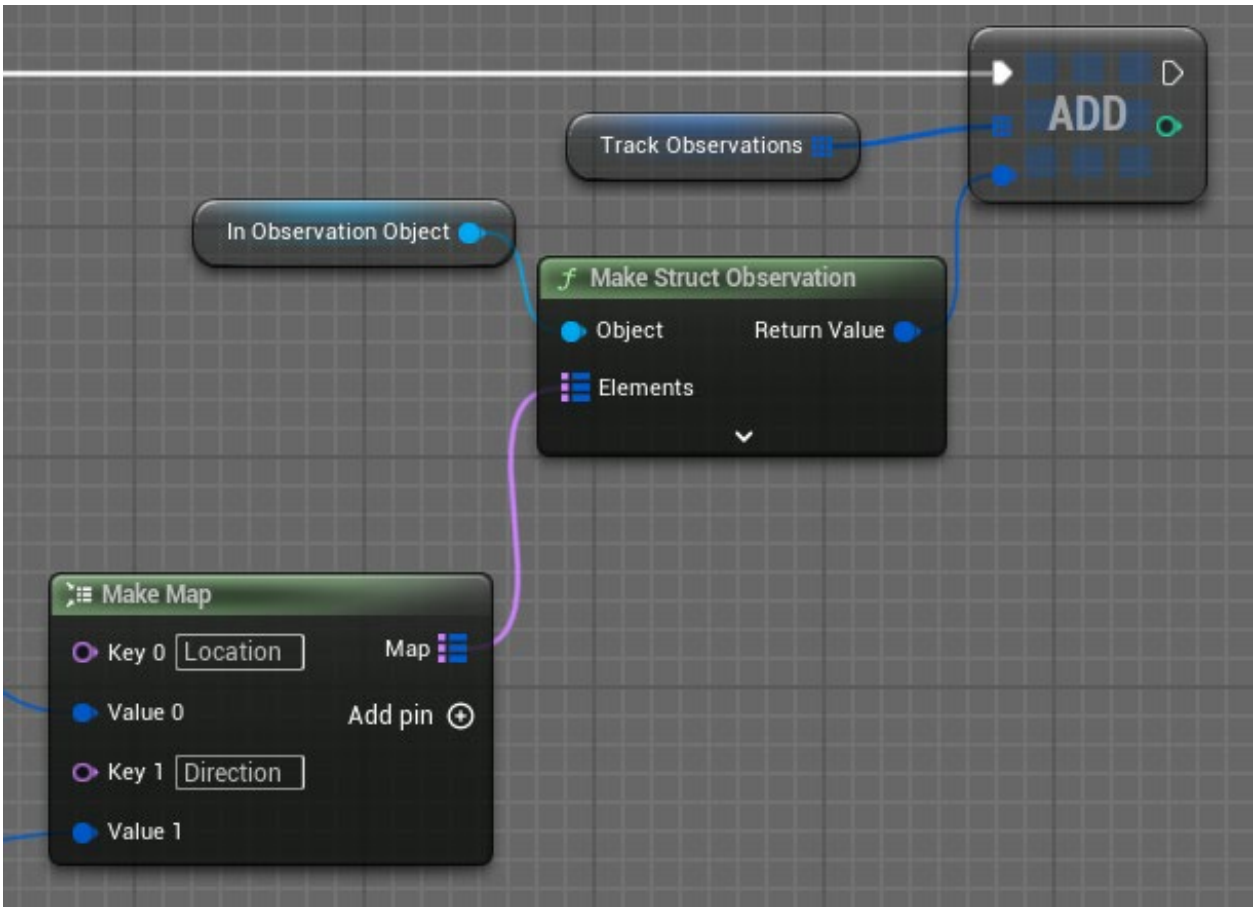


Figure B7: GatherAgentObservation function output

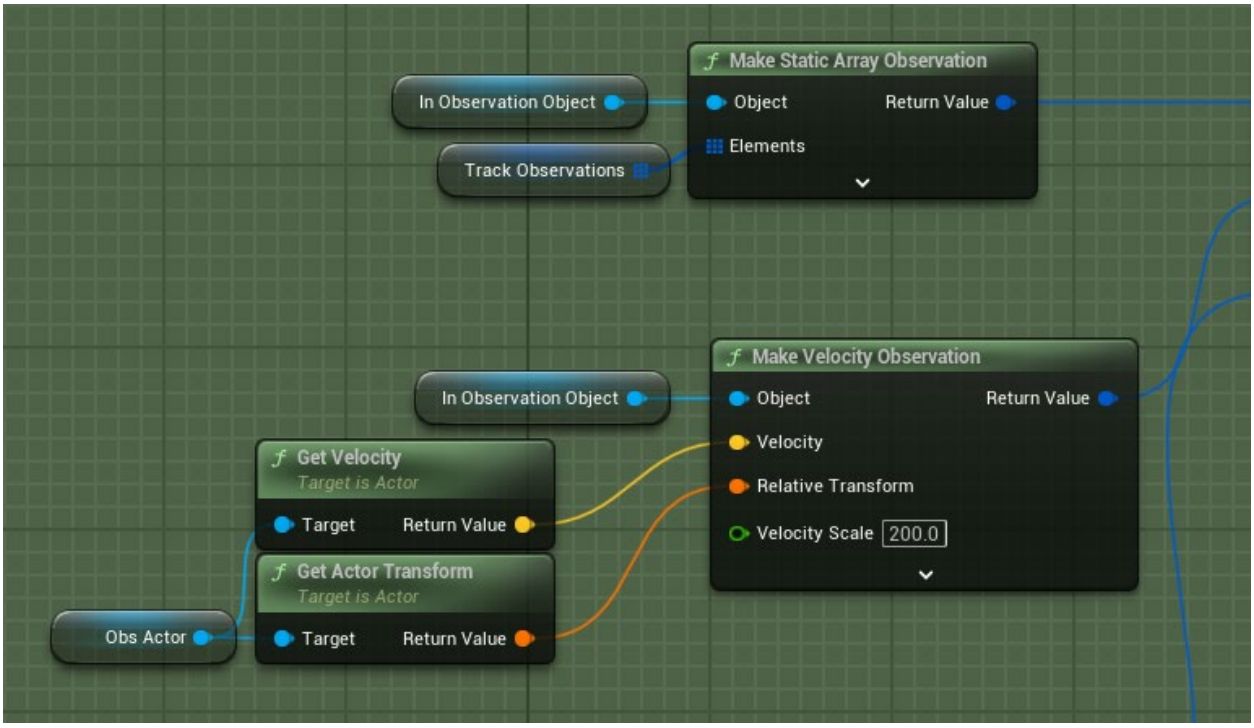


Figure B8: SpecifyAgentAction full implementation

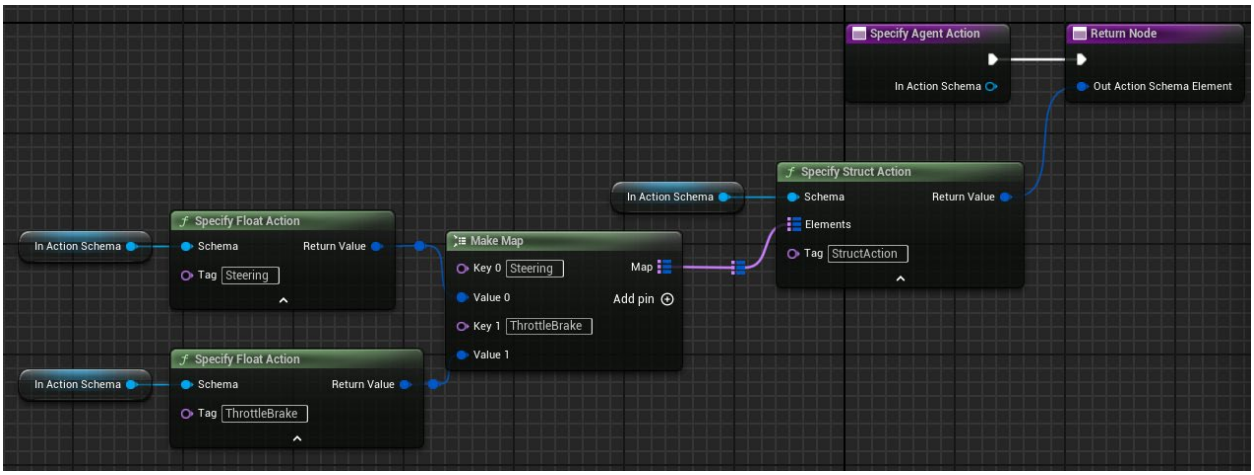


Figure B9: Throttle/brake action if the float action value is greater than 0

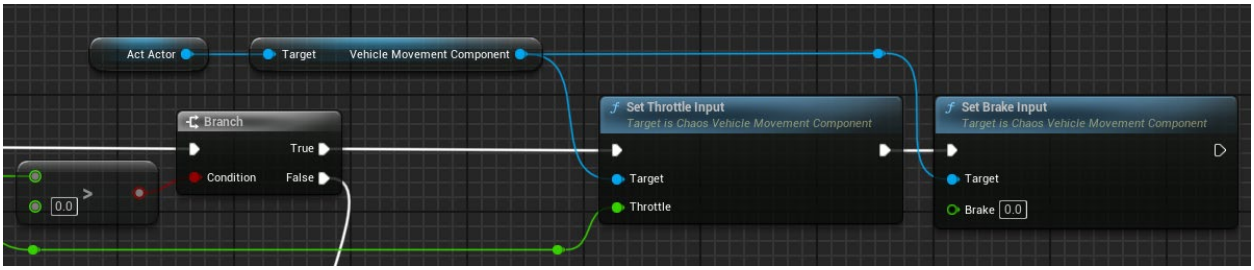


Figure B10: Throttle/brake action if the float action value is less than 0

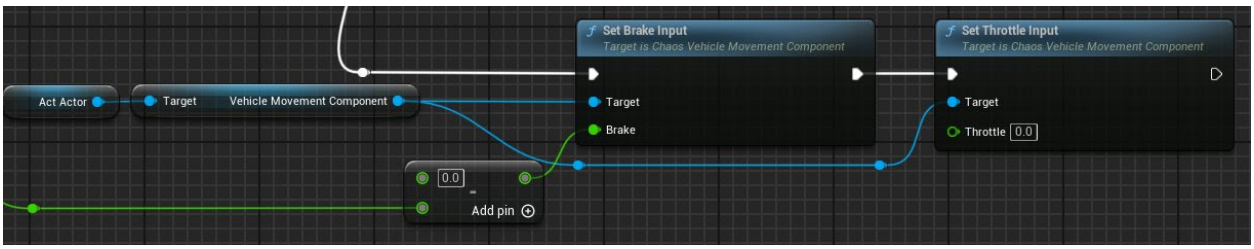


Figure B11: Accessing float action for Steering

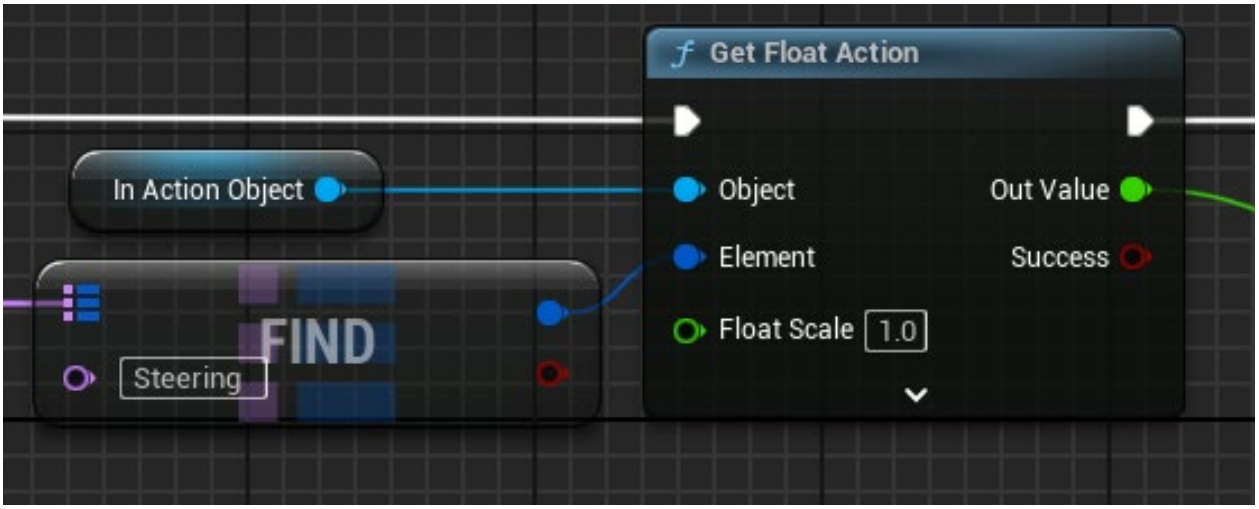


Figure B12: Accessing float action for Throttle/brake

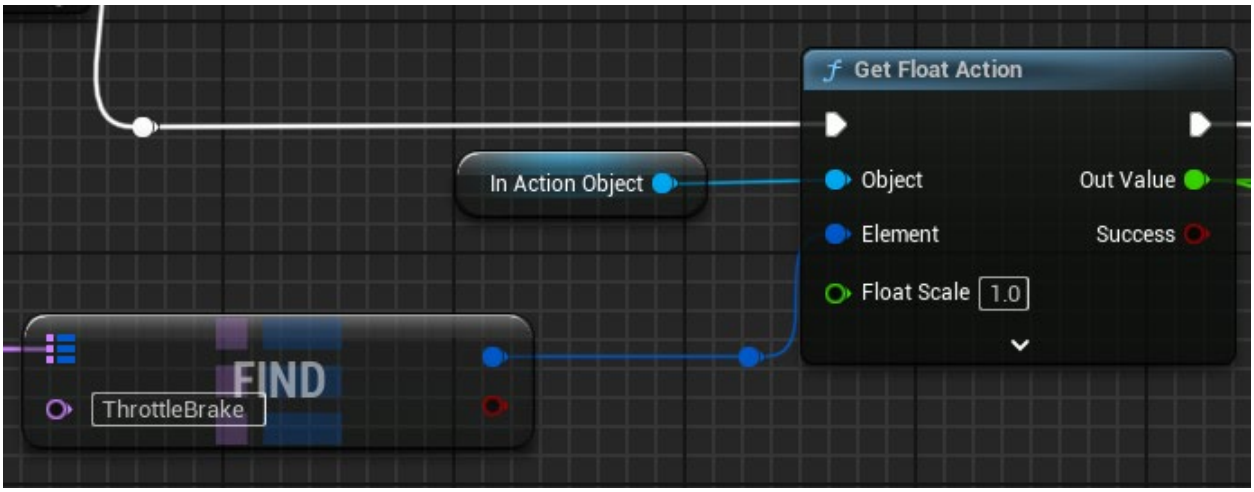


Figure B13: Penalty for leaving the road

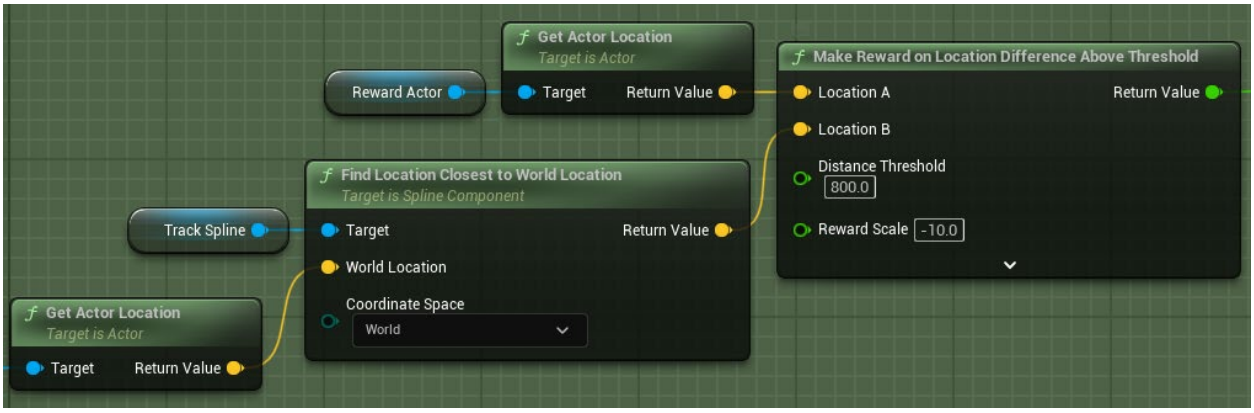


Figure B 14: Reward for driving as fast as possible along the spline

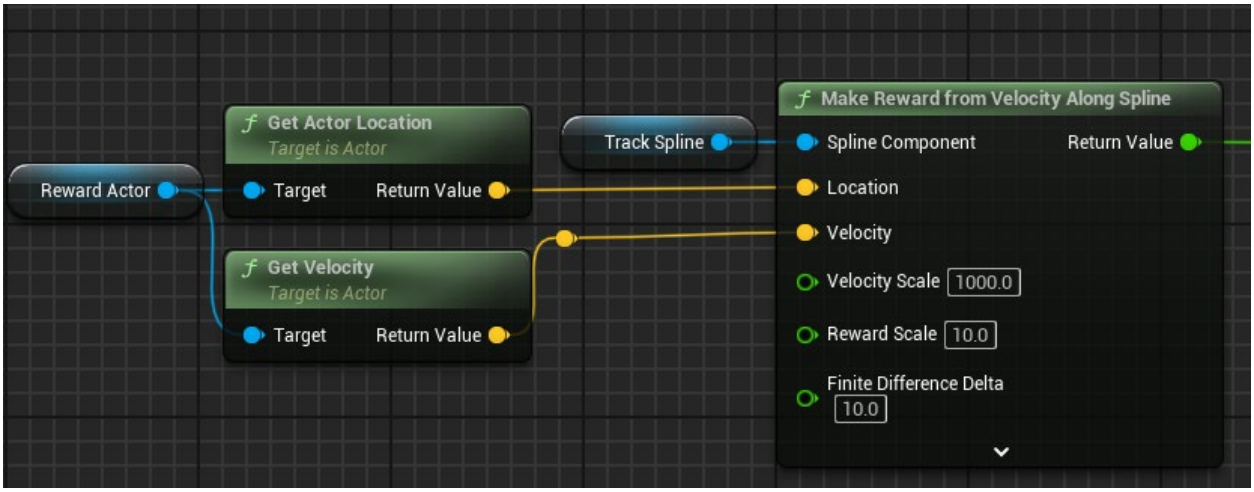
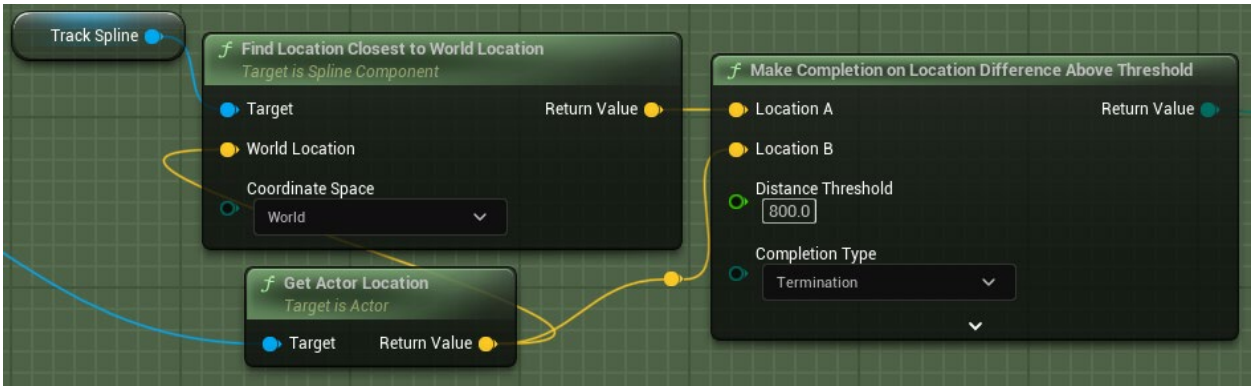


Figure B 15: Termination if the agent drives outside of the bounds



14.4 APPENDIX C: OBSTACLE AVOIDANCE – SETUP

Figure C1: Static obstacle used in this research

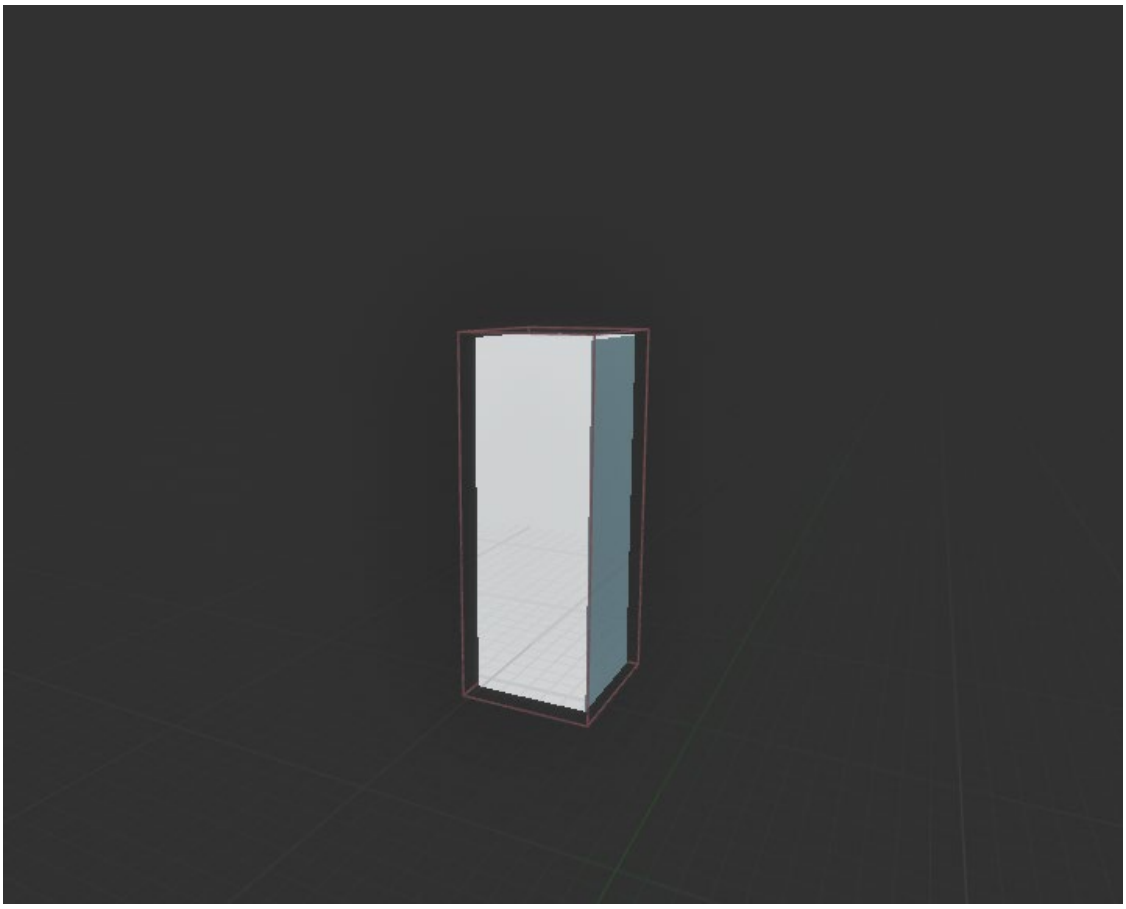


Figure C2: Randomizing the location of the obstacle

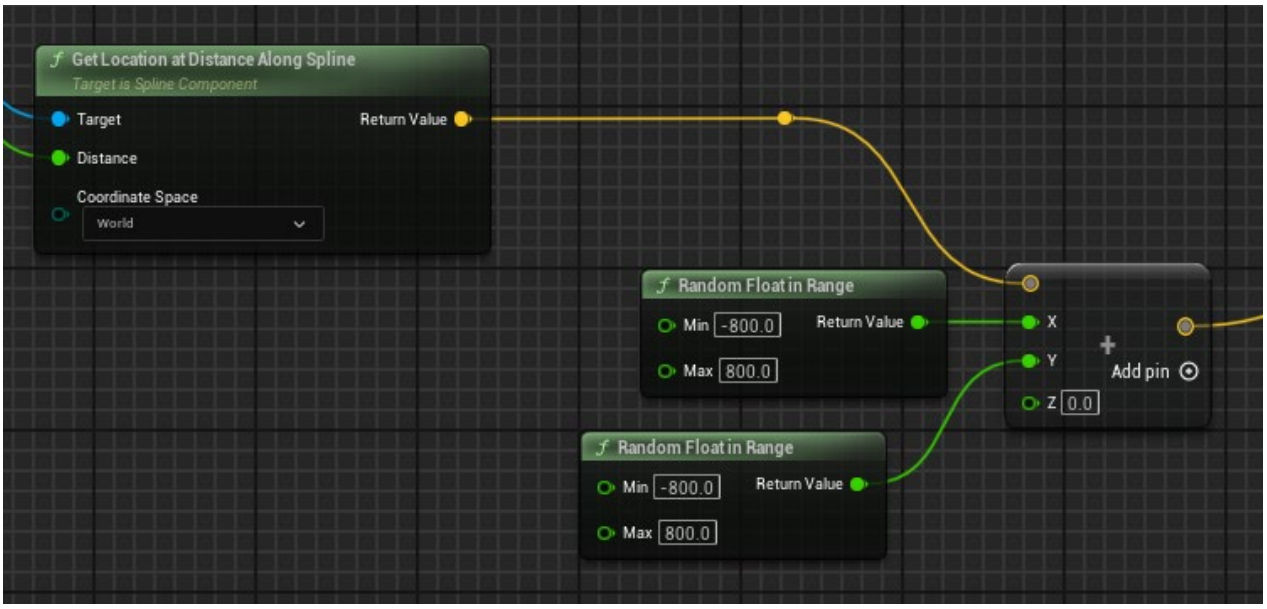
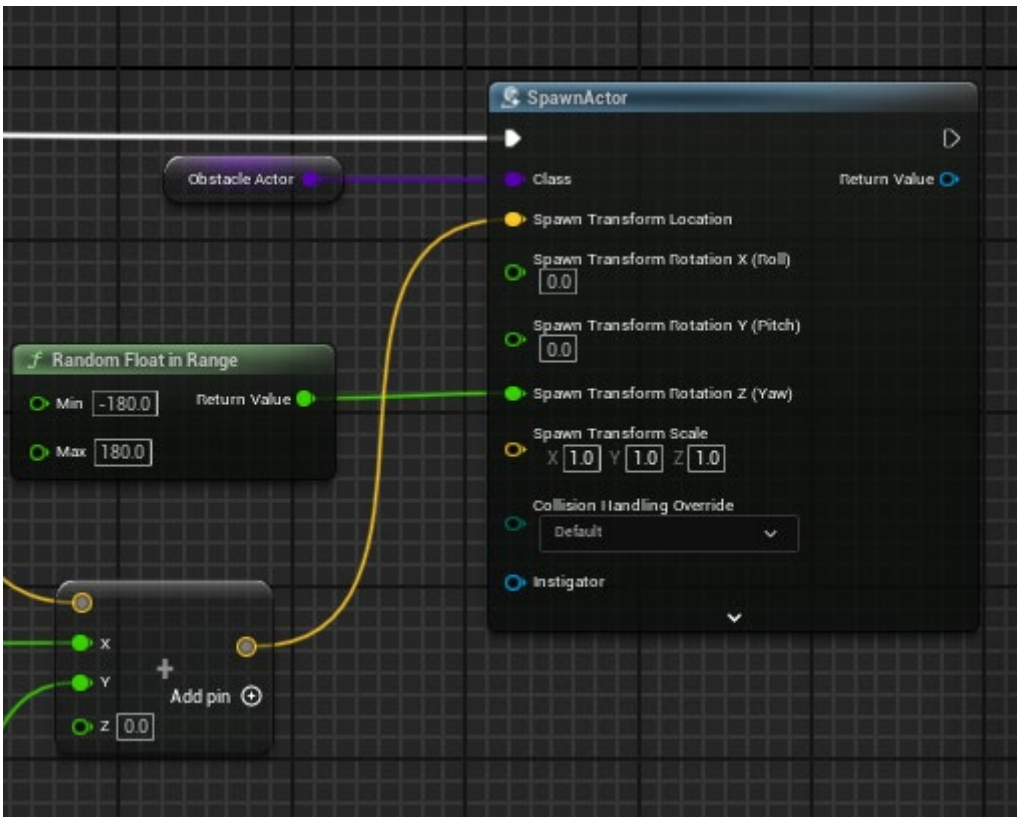


Figure C3: Spawning the obstacle



14.5 APPENDIX D: OBSTACLE AVOIDANCE – APPROACH 1

Figure D1: Calculating end of line segment for sphere tracing

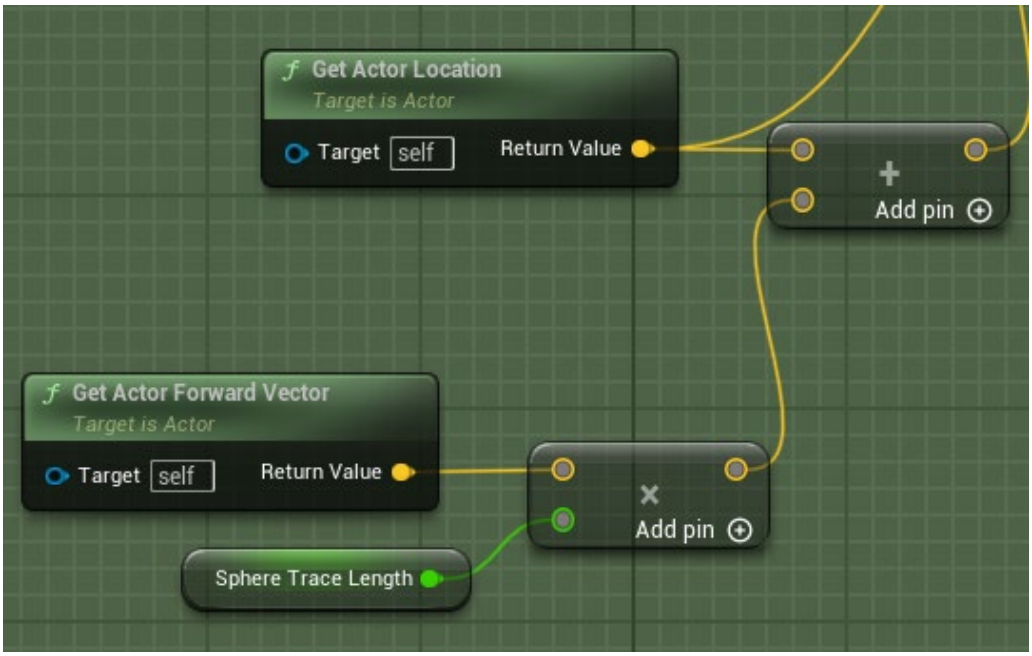


Figure D2: Sphere tracing for obstacles

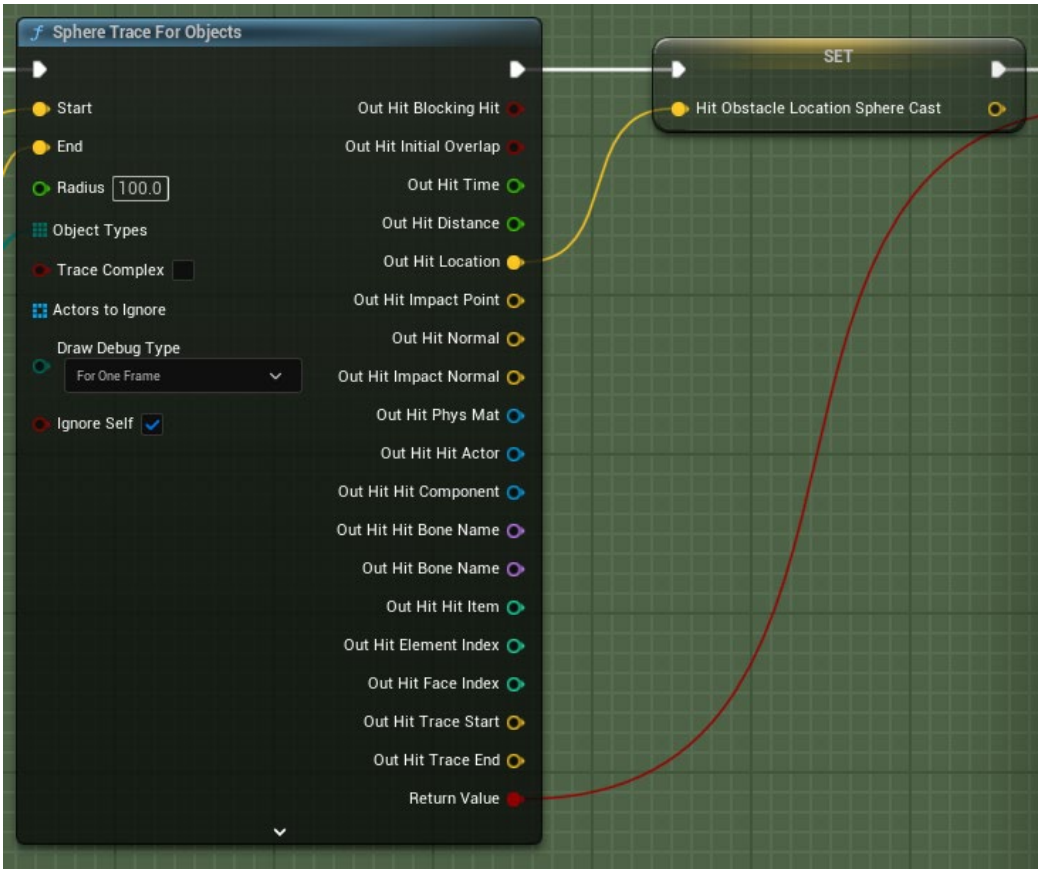


Figure D3: Gathering obstacle observation

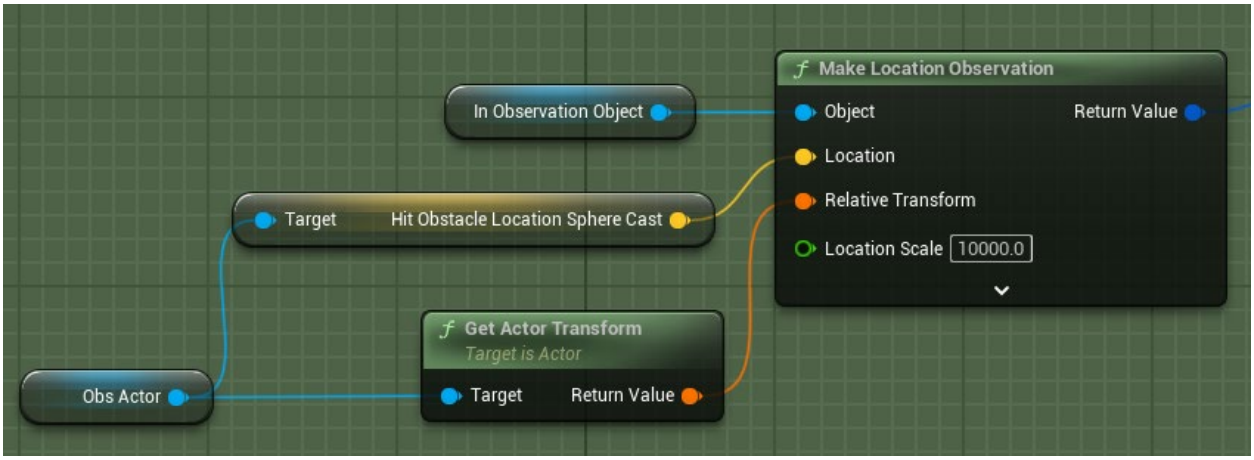


Figure D4: Specifying obstacle observation

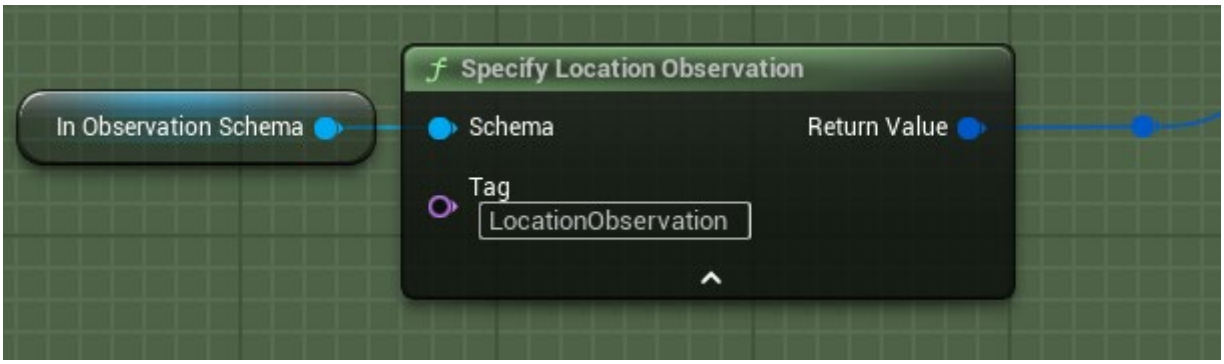


Figure D5: Calculating agent to obstacle vector

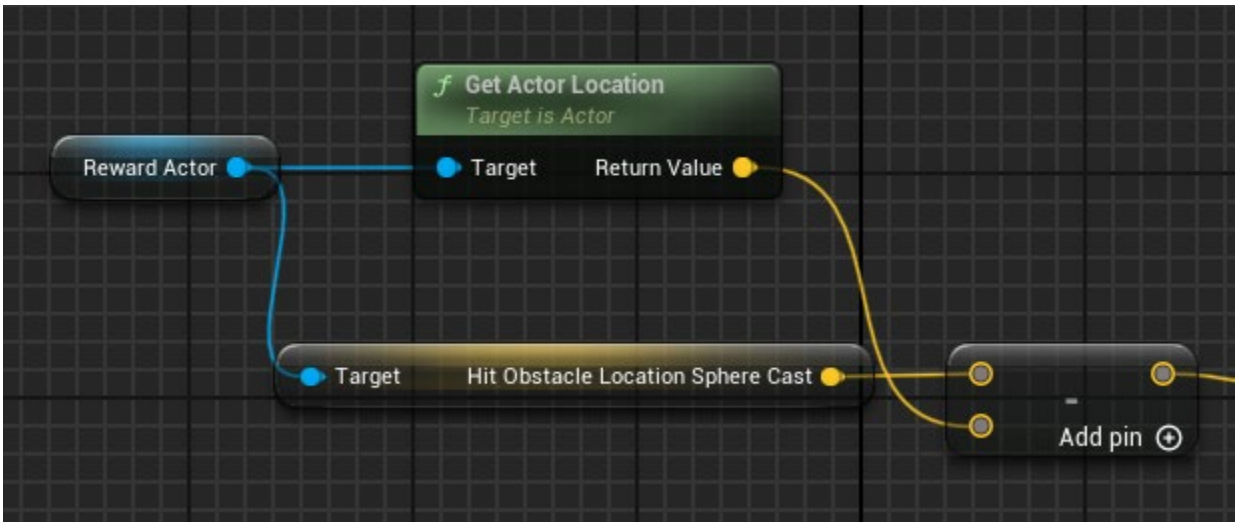
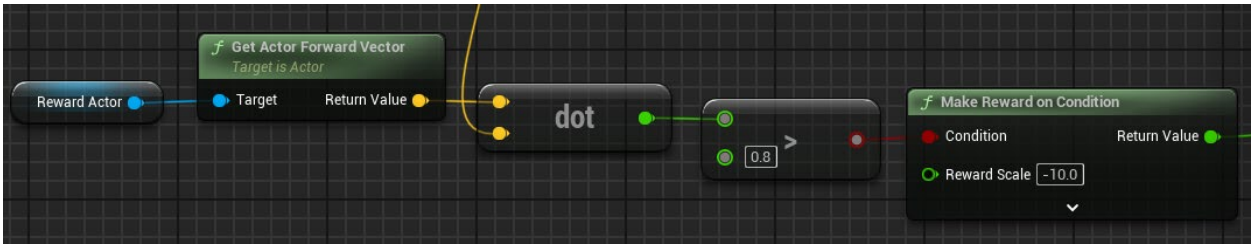


Figure D6: Agent penalty for facing an obstacle



14.6 APPENDIX E: OBSTACLE AVOIDANCE – APPROACH 2

Figure E1: Multi-sphere tracing for obstacles

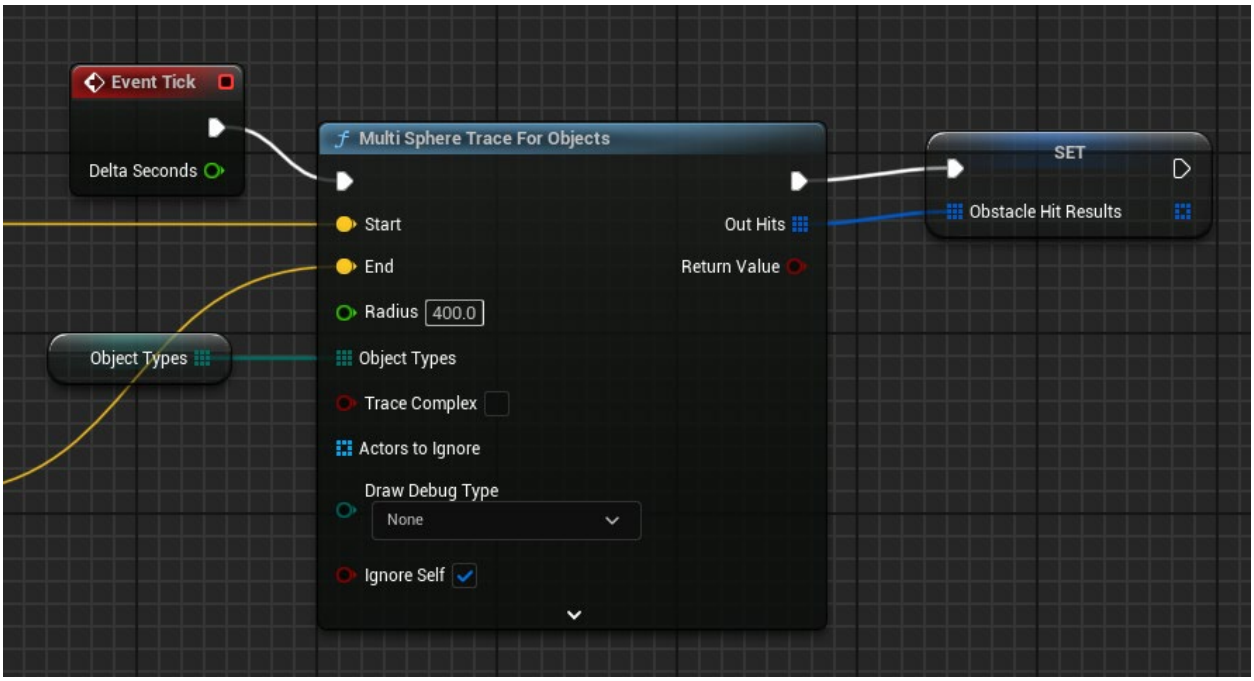


Figure E2: Specifying obstacle observation

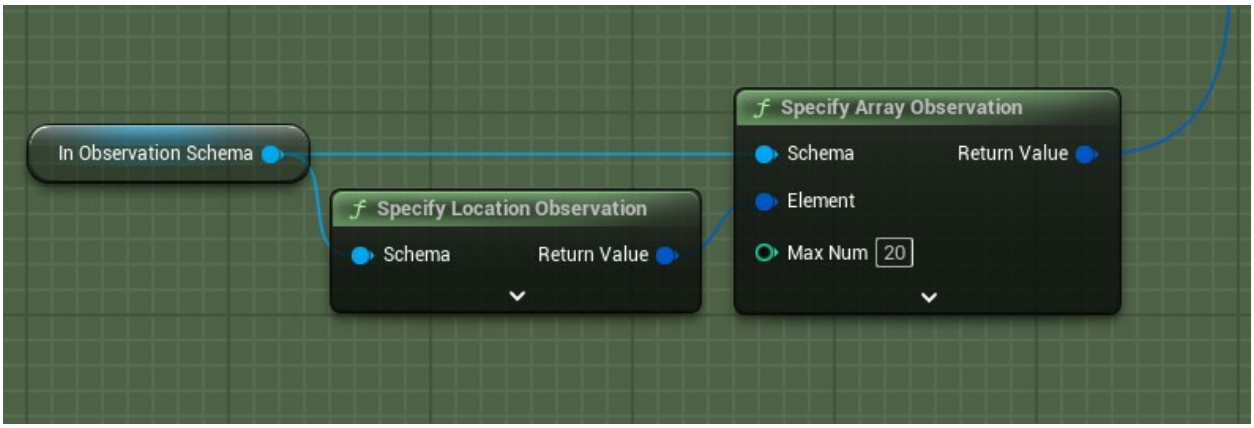


Figure E3: Gathering obstacle observation

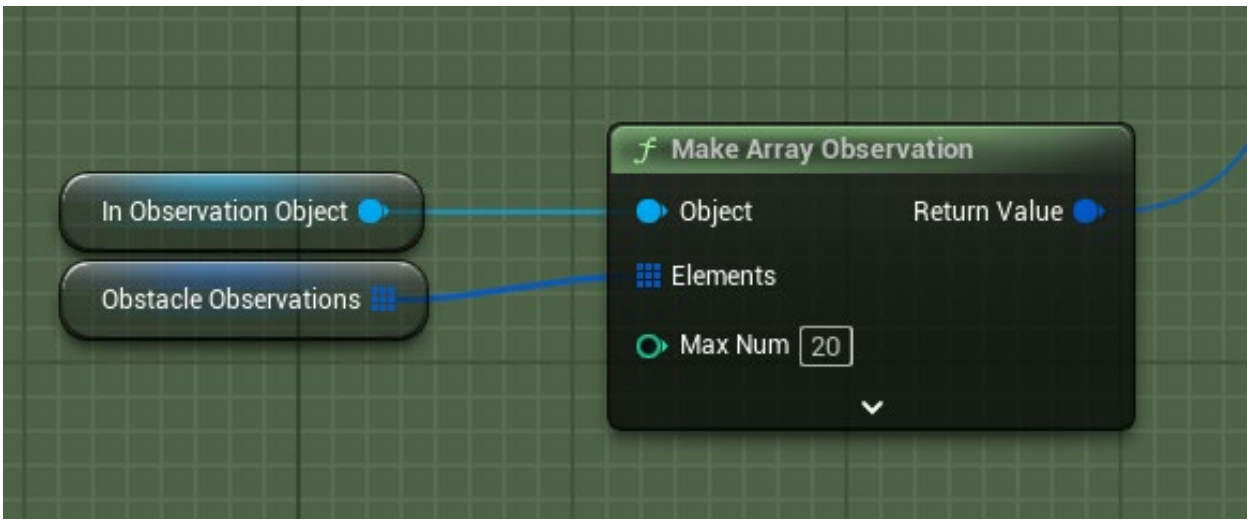
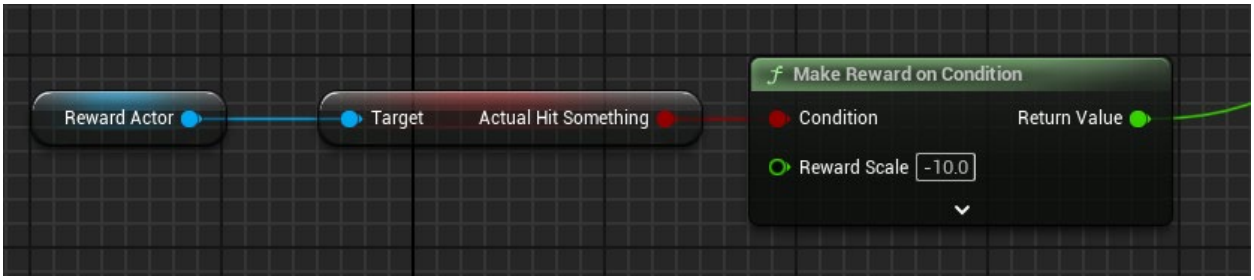


Figure E4: Agent penalty for colliding with an obstacle



14.7 APPENDIX F: OBSTACLE AVOIDANCE – APPROACH 3

Figure F1: Starting angles of agent's "vision"

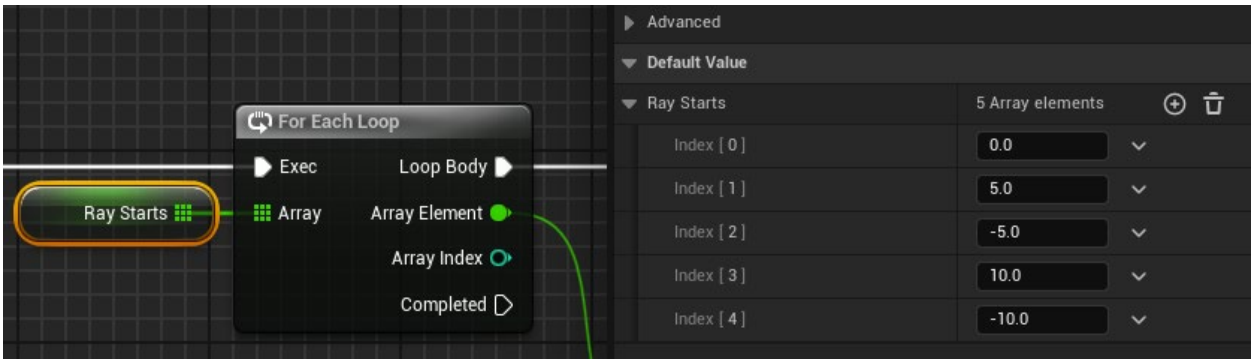


Figure F2: Rotating the forward vector to calculate "vision" vectors

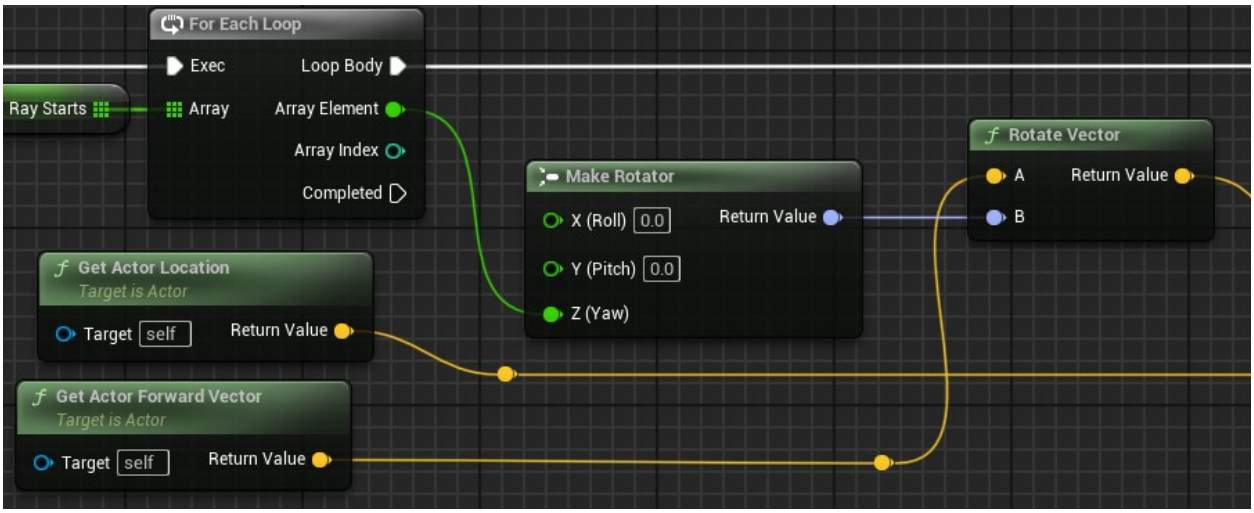


Figure F3: Simulating the "vision" using line trace

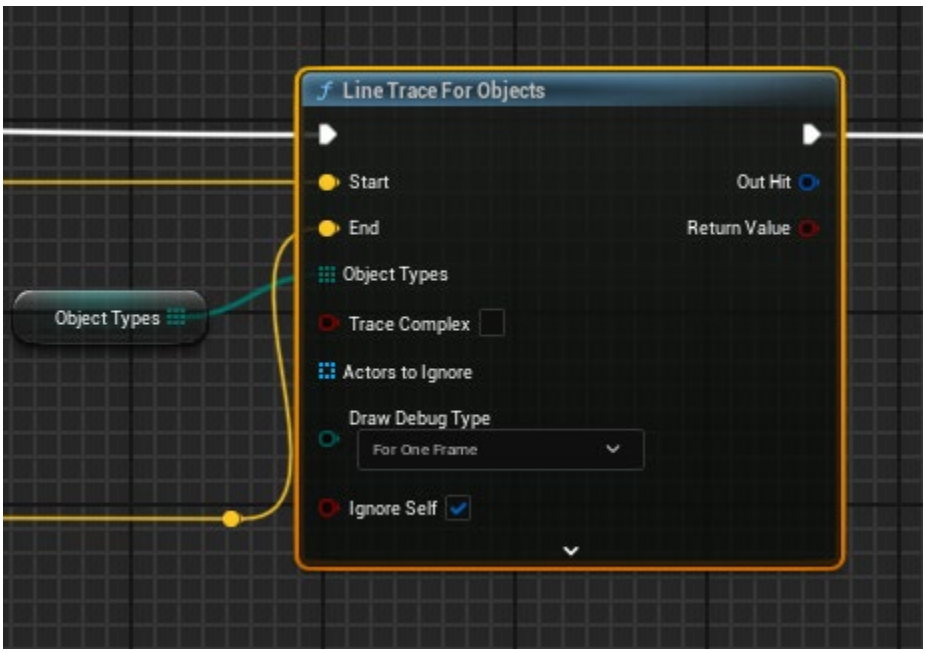


Figure F4: Specifying "vision" ray observation

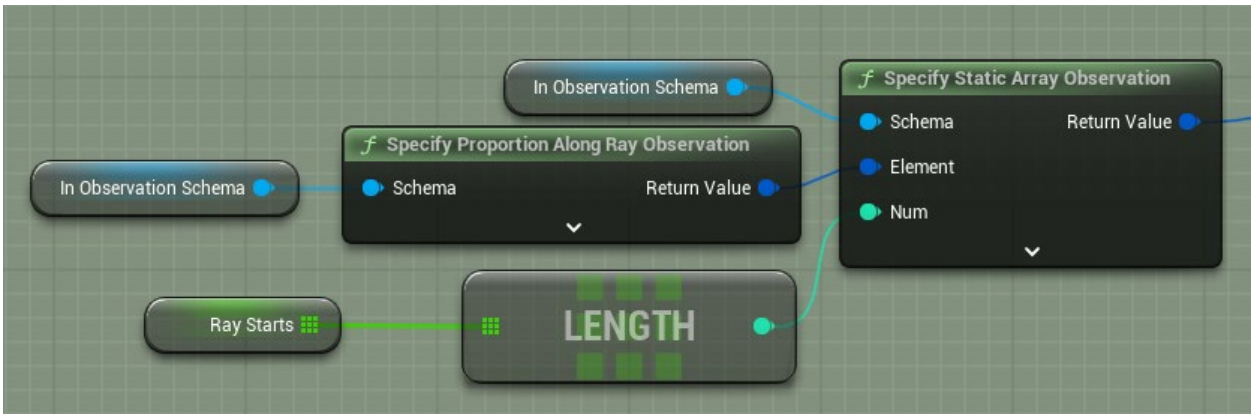


Figure F5: Specifying collision bool observation

Figure F6: Gathering "vision" ray observation

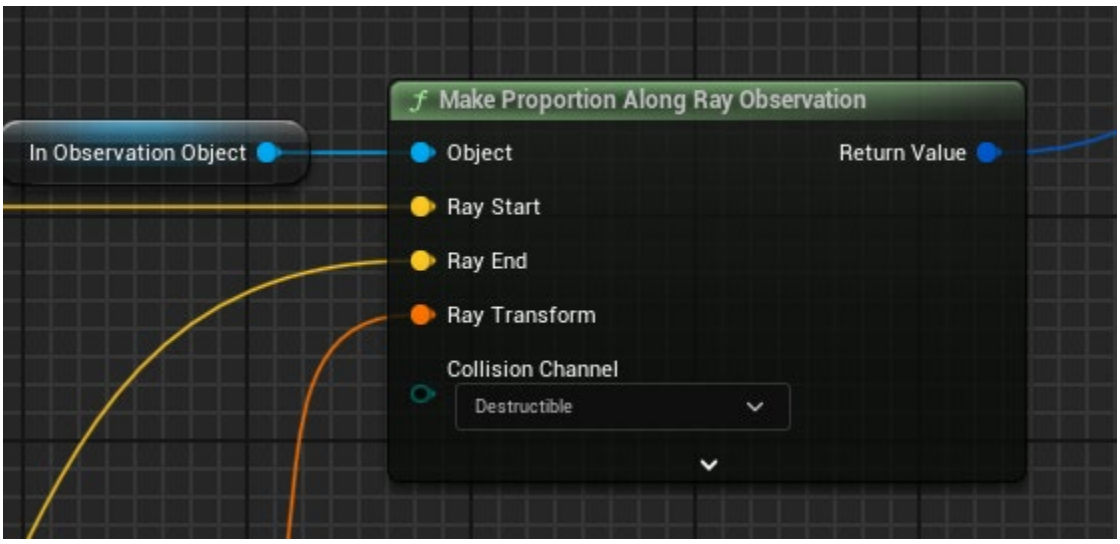


Figure F7: Gathering collision bool observation

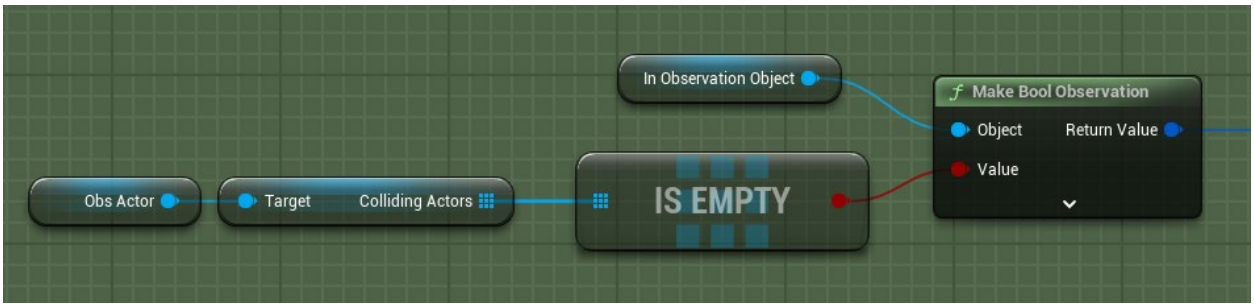


Figure F8: Negatively rewarding the agent for detecting obstacles

14.8 APPENDIX G: OBSTACLE AVOIDANCE – APPROACH 4

Figure G1: Calculating lines for vision at different angles

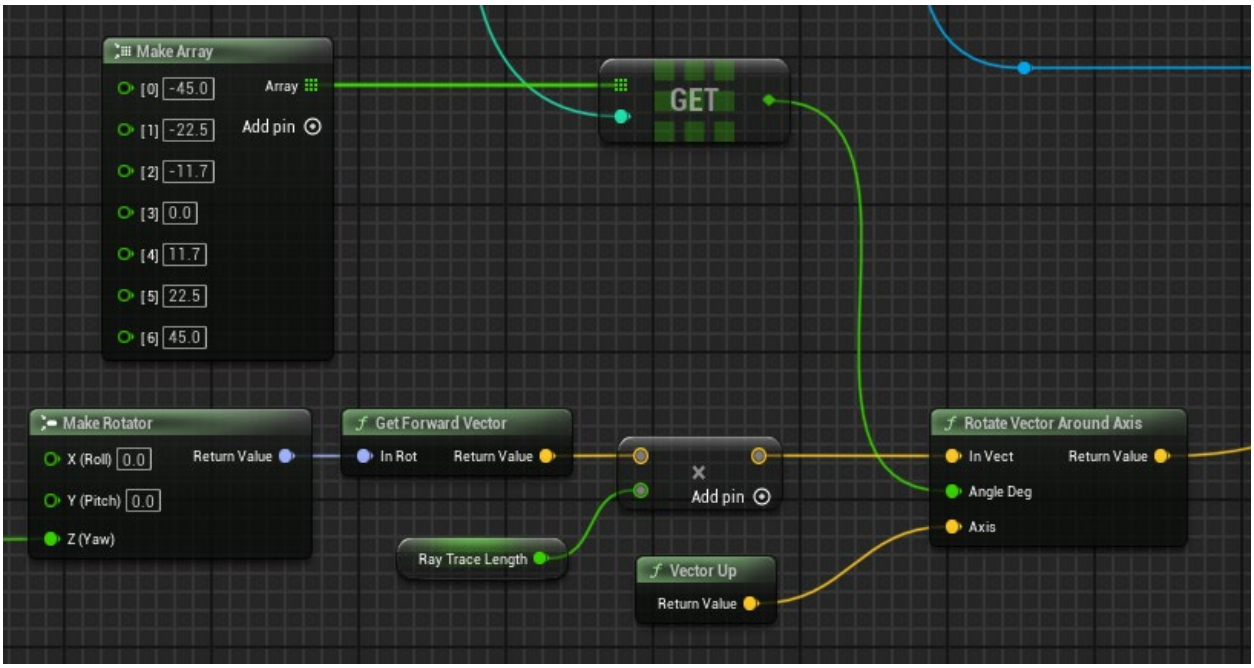


Figure G 2: Making ray observations, simulating the agent's vision

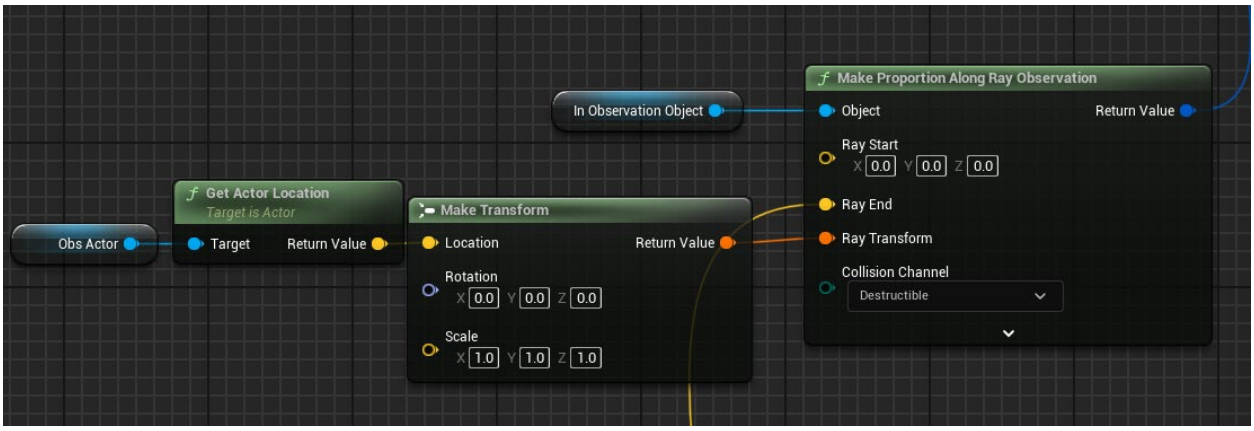


Figure G3: Observing too close obstacles that should cause a negative reward

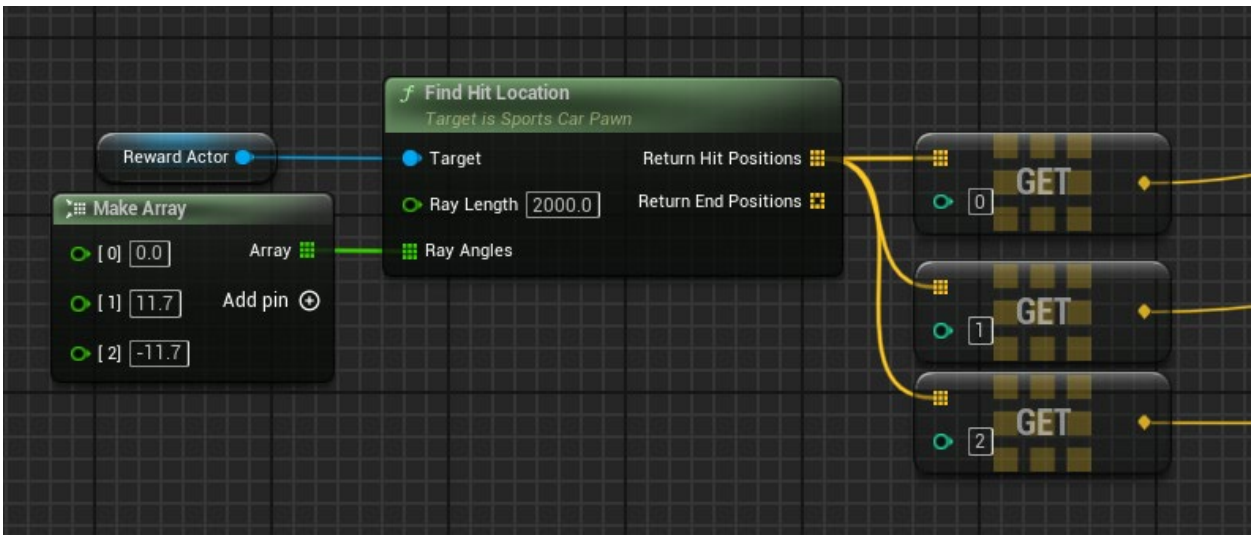


Figure G 4: Negative reward if the agent gets too close to an obstacle

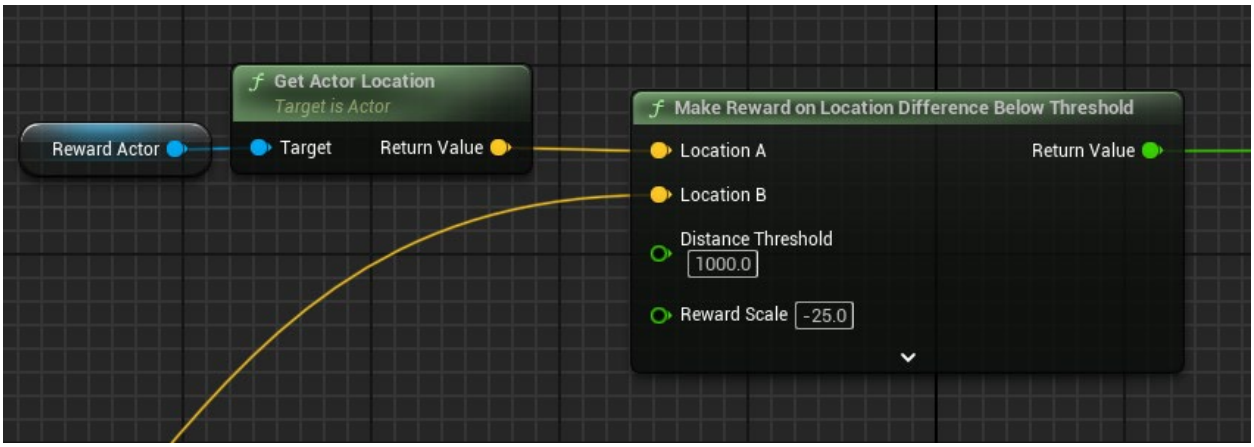
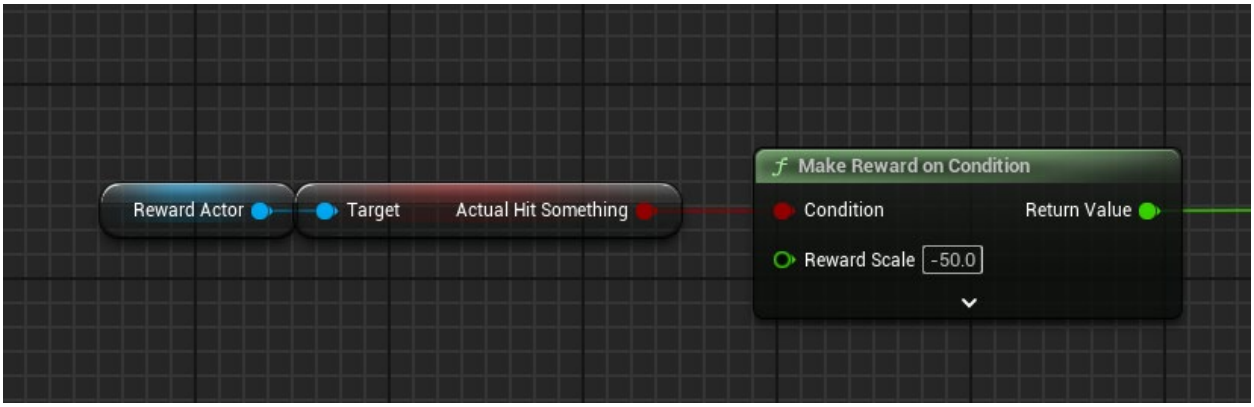


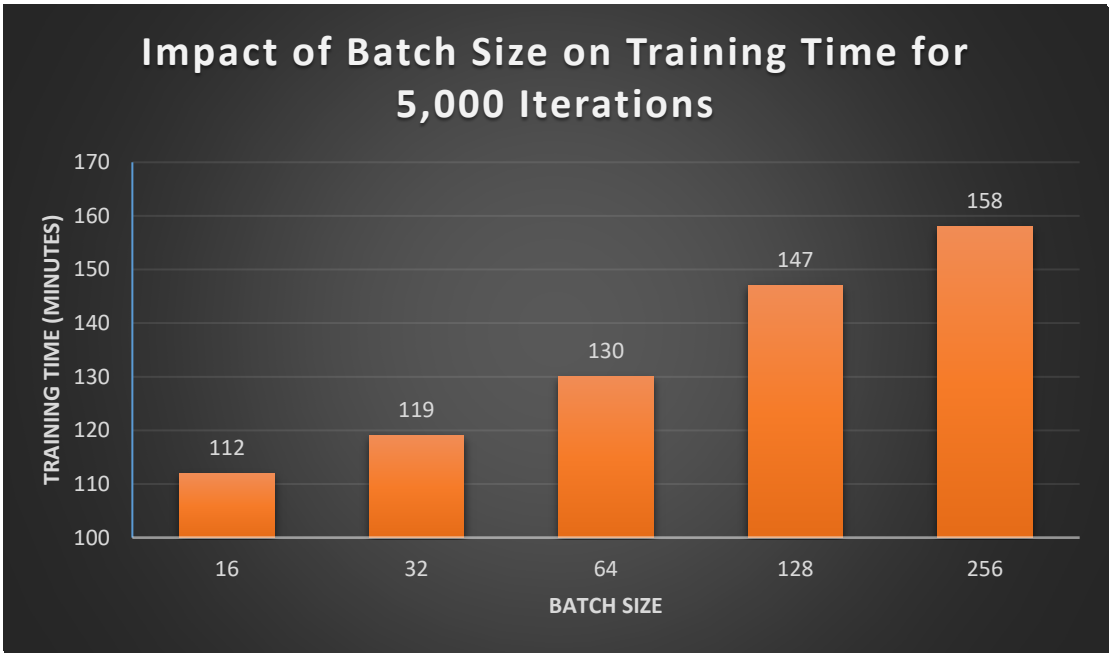
Figure G5: Negative reward for colliding with an obstacle



14.9 APPENDIX H: TRAINING RESULTS

Figure H1 demonstrates the relationship between batch size and training time for 5000 iterations. Larger batch sizes require more time to complete 5000 iterations. The training time increases gradually but stays consistent. Larger batch sizes process more data per iteration, increasing computational overhead and leading to longer training times, so this result was expected.

Figure H1: Training time vs batch size for 5000 iterations



The average return at 1000 iterations, as shown in Figure H2, indicates poor performance early in the training. Smaller batch sizes struggle the most, but none of the agents can deliver performant return value after 1000 iterations. At 5000 iterations, batch sizes of 16 and 32 demonstrate slightly improved behavior. The negative average return suggests that more training time is necessary for those batch sizes to see improvements. Batch size 64 shows marginal improvement, while batches 128 and 256 show significant improvement, indicating better learning performance compared to smaller batch sizes.

Larger batch sizes, particularly 256, demonstrate superior learning performance by the end of 5000 iterations, using the configuration described in Approach 4: Smart Agent. However, the slower progress in early iterations suggests that 5000 iterations are insufficient for smaller batch sizes.

Figure H2: Average return vs. batch size for 1000 and 5000 iterations